Laconic Branching Programs from the Diffie-Hellman Assumption*

Sanjam Garg[1], Mohammad Hajiabadi[2], Peihan Miao[3], and Alice Murphy[4]

[1] NTT Research and UC Berkeley, USA
sanjamg@berkeley.edu
[2] University of Waterloo, Canada
mdhajiabadi@uwaterloo.ca
[3] Brown University, USA
peihan_miao@brown.edu
[4] University of Waterloo, Canada
anlmurph@uwaterloo.ca

Abstract. Laconic cryptography enables secure two-party computation (2PC) on unbalanced inputs with asymptotically-optimal communication in just two rounds of communication. In particular, the receiver (who sends the first-round message) holds a long input and the sender (who sends the second-round message) holds a short input, and the size of their communication to securely compute a function on their joint inputs only grows with the size of the sender's input and is independent of the receiver's input size. The work on laconic oblivious transfer (OT) [Cho et al. CRYPTO 2017] and laconic private set intersection (PSI) [Alamati et al. TCC 2021] shows how to achieve secure laconic computation for OT and PSI from the Diffie-Hellman assumption.

In this work, we push the limits further and achieve laconic branching programs from the Diffie-Hellman assumption. In particular, the receiver holds a large branching program P and the sender holds a short input x. We present a two-round 2PC protocol that allows the receiver to learn x iff $P(x) = 1$, and nothing else. The communication only grows with the size of x and the depth of P, and does not further depend on the size of P.

Keywords: Laconic cryptography, unbalanced secure computation, branching programs

1 Introduction

Suppose a server holds a *large* set of elements Y (which could be exponentially large) that can be represented as a polynomial-sized branching program P, that is, $y \in Y$ iff $P(y) = 1$. The server would like to publish a *succinct* digest of Y such that any client who holds a *small* set X can send a *short* message to the server to allow her to learn the set intersection $X \cap Y$ but nothing beyond that.

This is a special case of the secure two-party computation (2PC) [Yao86] problem, where two mutually distrustful parties, each holding a private input x and y respectively, would like to jointly compute a function f over their private inputs without revealing anything beyond the output of the computation. Garbled circuits [Yao86] together with oblivious transfer (OT) [Rab81, Rab05] enables 2PC for any function f with two rounds of communication: one message from the *receiver* to the *sender* and another message from the sender back to the receiver. This approach achieves the optimal round complexity; nevertheless, it requires the communication complexity to grow with the size of f. In particular, if we represent f as a Boolean circuit, then the communication grows with the number of gates in the circuit, which grows at least with the size of the inputs x and y. For unbalanced input lengths (i.e., $|x| \gg |y|$ or $|x| \ll |y|$), *is it possible to make the communication only grow with the shorter input and independent of the longer input?*

Long Sender Input. When the sender has a long input, i.e. $|x| \gg |y|$, we can use fully homomorphic encryption (FHE) [Gen09] to achieve communication that only grows with the receiver's input length $|y|$ plus the output length. This technique works for any function but can only be based on variants of the learning with errors (LWE) assumption [GSW13]. For simpler functions that can be represented by a branching program, in particular, if the sender holds a private large branching program P and the receiver holds a private short input y, the work of Ishai and Paskin [IP07] illustrates how to construct 2PC for $P(y)$ where the communication

only grows with $|y|$ and the *depth* of P, and does not further depend on the size of P. Their construction is generic from a primitive called *rate-1 OT*, which can be built based on a variety of assumptions such as DCR, DDH, QR, and LWE assumptions with varying efficiency parameters [IP07, DGI+19, GHO20, CGH+21]. In this setting, there are works in secure BP evaluation for applications in machine learning and medicine [BPSW07, BFK+09, KNL+19, CDPP22]. Our results concern the dual setting, in which the receiver has the longer input and is the party that learns the output. Moreover, this should be achieved in only two rounds of communication.

Long Receiver Input. When the receiver has a long input, i.e. $|x| \ll |y|$, a recent line of work on *laconic cryptography* [CDG+17, QWW18, DGGM19, ABD+21, ALOS22] focuses on realizing secure 2PC with asymptotically-optimal communication in two rounds. In particular, the receiver has a large input and the size of her protocol message only depends on the security parameter and not her input size. The second message (sent by the sender) as well as the sender's computation may grow with the size of the sender's input, but should be independent of the receiver's input size.

In this dual setting, the work of Quach, Wee, and Wichs [QWW18] shows how to realize laconic 2PC for general functionalities using LWE. Regarding laconic 2PC for simpler functions from assumptions other than LWE, much less is known compared to the setting of long sender inputs.

The work of Cho et al. [CDG+17] introduced the notion of laconic oblivious transfer (laconic OT), where the receiver holds a large input $D \in \{0,1\}^n$, the sender holds an input $(i \in [n], m_0, m_1)$, and the two-round protocol allows the receiver to learn $(i, m_{D[i]})$ and nothing more. The communication complexity as well as the sender's computation only grow with the security parameter and is independent of the size of D. Besides LWE [QWW18], laconic OT can be built from DDH, CDH, and QR [CDG+17, DG17].[5] Recent work [ABD+21, ALOS22] extends the functionality to laconic private set intersection (laconic PSI), where the sender and receiver each holds a private set of elements X and Y respectively ($|X| \ll |Y|$), and the two-round protocol allows the receiver to learn the set intersection $X \cap Y$ and nothing more. The communication complexity and the sender's computational complexity are both independent of $|Y|$. Laconic PSI can be built from CDH/LWE [ABD+21] or pairings [ALOS22].

Both laconic OT and laconic PSI can be viewed as special cases of a branching program. Recall that in the setting of long sender input, where a sender has a large branching program, we have generic constructions from rate-1 OT, which can be built from various assumptions. However, in the dual setting of long receiver input, we no longer have such a generic construction. Laconic OT *seems* to be a counterpart building block in the dual setting, but it does *not* give us laconic branching programs. Given the gap between the two settings, we ask the following question:

Can we achieve laconic branching programs from assumptions other than LWE?

This diversifies the set of assumptions from which laconic MPC can be realized. It also increases our understanding of how far each assumption allows us to expand the functionality, which helps in gaining insights into the theoretical limits of the assumptions themselves.

1.1 Our Results

We answer the above question in the affirmative. In particular, as a natural counterpart to the aforementioned setting of long sender input, when the receiver holds a private large branching program BP and the sender holds a private short input x, we construct a two-round 2PC protocol allowing the receiver to learn x iff $\mathrm{BP}(x) = 1$, and nothing else. The communication only grows with $|x|$ and the *depth* of BP, and does not further depend on the size of BP. Furthermore, the sender's computation also only grows with $|x|$ and the depth of BP. The receiver's computation grows with the number of BP nodes and the number of root-to-leaf paths. Our construction is based on anonymous hash encryption schemes [BLSV18], which can in turn be based on CDH/LWE [DG17, BLSV18].

[5] Importantly, in laconic OT, the receiver's second-phase computation time should have at most a polylog dependence on $|D|$. This can be achieved in the laconic OT setting because the index i is known to the receiver. In other settings, such as laconic PSI, this cannot be realized (without pre-processing) because *not* probing a particular database entry leaks information about the sender's input.

Sender Security. We achieve what we call *weak sender security* which says if $\mathrm{BP}(x) = 0$, then no information about x is leaked; else, there are no privacy guarantees for x. A stronger security requirement would be that in the latter case, the receiver only learns $\mathrm{BP}(x)$, and no other information about x. Unfortunately, realizing strong sender security is too difficult in light of known barriers: it generically implies a notion called *private laconic OT* [CDG+17, DGI+19]. Private laconic OT is laconic OT in which the index i chosen by the sender is also kept hidden from the receiver. The only existing construction of private laconic OT with polylogarithmic communication uses techniques from laconic secure function evaluation and is based on LWE [QWW18]. In particular, it is not known if private laconic OT can be realized using Diffie-Hellman assumptions.

Strong sender security allows one to achieve *laconic PSI cardinality*, a generalization of laconic PSI. In the PSI cardinality problem, the receiver learns only the size of the intersection and nothing about the intersection set itself. Strong sender security for a receiver with a large set S and a sender with a single element x would allow the receiver to *only* learn whether or not $x \in S$. This immediately implies laconic PSI cardinality by having the sender send a second-round protocol message for each element in its set. Laconic PSI cardinality generically implies private laconic OT, establishing a barrier. The same barriers prevented [DKL+23] from building laconic PSI cardinality. We can get laconic PSI as an application of our results (and other applications discussed below), but our results do not allow us to realize laconic PSI cardinality. More specifically, after receiving the second-round message from the sender, the receiver in our protocol works by checking which path in their BP tree (if any) decrypts to "accept". If there is an accepting path, then $\mathrm{BP}(x) = 1$, where x is the sender's input. But this reveals the value of x since the receiver knows which path resulted in acceptance.

Applications. Our laconic branching program construction directly implies laconic OT and laconic PSI, as their functionalities can be represented as branching programs. Moreover, we can capture other functionalities not considered by previous work, such as private-set unions (PSU). A branching program for PSU can be obtained by making local changes to a branching program for PSI. (See Section 5.) This demonstrates the versatility of our approach, giving a unifying construction for all these functionalities. In contrast, the accumulator-based PSI constructions in [ABD+21, ALOS22, DKL+23] are crucially tied to the PSI setting, and do not seem to extend to the PSU setting. This is because the sender's message to the receiver only provides enough information to indicate which element (if any) in the receiver's set is also held by the sender. In essence, only the index of this element within the receiver's set needs to be conveyed in the sender's message. In the PSU setting, on the other hand, there could be elements in the union that do not exist in the receiver's set. So, the sender's message needs to contain more information than an index. If the sender's element is not in the receiver's set, the receiver needs to be able to recover the sender's element from the message.

Our techniques can be used in unbalanced PSI where the receiver holds a large set (possibly of exponential size) that can be represented as a branching program. For instance, a recent work by Garimella et al. [GRS22] introduced the notion of *structure-aware PSI* where one party's (potentially large) set Y is publicly known to have a certain structure. As long as the publicly known structure can be represented as a branching program, our techniques can be used to achieve a two-round fuzzy-matching PSI protocol where the communication only grows with the size of the smaller set $|X|$ and the *depth* of the branching program, and does not further depend on $|Y|$, which could potentially be *exponentially* large.

2 Technical Overview

Our constructions are based on hash encryption (HE) schemes [DG17, BLSV18]. An HE scheme, parameterized by $n = n(\lambda)$ (where λ is the security parameter), consists of a hash function $\mathsf{Hash}(\mathsf{pp}, \cdot) : \{0,1\}^n \to \{0,1\}^\lambda$ and associated HEnc and HDec functions. One can encrypt n pairs of plaintexts $\mathbf{m} := (m_{i,b})$ (for $i \in [n]$ and $b \in \{0,1\}$) with respect to $h := \mathsf{Hash}(\mathsf{pp}, z)$ to get $\mathsf{cth} \leftarrow \mathsf{HEnc}(h, \mathbf{m})$.[6] The ciphertext cth is such that given z, one may recover $(m_{1,z[1]}, \ldots, m_{n,z[n]})$ from cth, while maintaining semantic security for $(m_{i,1-z_i})$ even in the presence of z. HE can be realized using CDH/LWE [DG17, BLSV18].

Consider a simple example where the receiver R has a depth-one BP on bits (see Def. 3 for branching programs) where the root node encodes index $i^* \in [n]$ and its left child encodes accept ($b_0 := \mathsf{accept}$) and

[6] HEnc also takes the public parameter pp as input, but we omit it here for brevity.

its right child encodes reject ($b_1 := \mathsf{reject}$). This BP evaluates an input x by checking the bit value at index i^*. If $x[i^*] = 0$, then the value of the left child is output: $b_0 = \mathsf{accept}$. If $x[i^*] = 1$, then the value of the right child is output: $b_1 = \mathsf{reject}$. As a starting point, suppose R only wants to learn if $\mathsf{BP}(x) = 1$, where x is the sender's input. The receiver hashes $h := \mathsf{Hash}(\mathsf{pp}, (i^*, b_0, b_1))$, padding the input if necessary, and sends h to the sender, S. S has the following circuit $\mathsf{F}[x]$ with their input x hardwired: on input (j, q_0, q_1), $\mathsf{F}[x]$ outputs $q_{x[j]}$. S garbles $\mathsf{F}[x]$ to get a garbled circuit $\widetilde{\mathsf{F}}$ and corresponding labels $(\mathsf{lb}_{i,b})$. S uses the hash value, h, from R to compute $\mathsf{cth} \leftarrow \mathsf{HEnc}(h, (\mathsf{lb}_{i,b}))$. Finally, S sends $(\widetilde{\mathsf{F}}, \mathsf{cth})$ to R. The receiver, given her hash pre-image value $z := (i^*, \mathsf{accept}, \mathsf{reject})$ can only recover $(\mathsf{lb}_{i,z[i]})$, allowing her in turn to learn $\mathsf{F}[x](z)$ from the garbled circuit, outputting either accept ($\mathsf{BP}(x) = 1$) or reject ($\mathsf{BP}(x) = 0$).

Beyond depth 1. Next, consider the BP of depth 2 in Fig. 1 held by the receiver, R. Each internal node encodes an index, $\mathsf{root}, \mathsf{left}, \mathsf{right} \in [n]$. The four leaves have values with variables $(b_{00}, b_{01}, b_{10}, b_{11})$. For $i, j \in \{0, 1\}$, $b_{ij} \in \{\mathsf{accept}, \mathsf{reject}\}$. Suppose $x[\mathsf{root}] = 0$, where x is the sender's input, so the root-leaf path induced by $\mathsf{BP}(x)$ first goes left. If the sender, S, 'by some miracle' knows the hash value $h_0 := \mathsf{Hash}(\mathsf{pp}, (\mathsf{left}, b_{00}, b_{01}))$, he can, as above, send a garbled circuit for $\mathsf{F}[x]$ and an HE ciphertext wrt h_0 of the underlying labels, allowing R

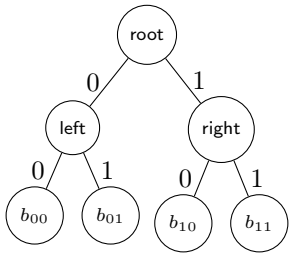

Fig. 1: Depth 2 BP example

to evaluate $\mathsf{F}[x](\mathsf{left}, b_{00}, b_{01})$. But S does not know the value of h_0, nor does he know whether the first move is left or right, because the BP is hidden from S. Moreover, R cannot send both $h_0 := \mathsf{Hash}(\mathsf{pp}, (\mathsf{left}, b_{00}, b_{01}))$ and $h_1 := \mathsf{Hash}(\mathsf{pp}, (\mathsf{right}, b_{10}, b_{11}))$ because (a) there will be a size blowup (the communication will grow with the size, and not the depth, of the BP), and (b) R will learn more information than necessary because S does not know *a priori* whether the induced computation travels left or right, so he has to encrypt the labels under both h_0 and h_1. But encrypting the labels $(\mathsf{lb}_{i,b})$ under both h_0 and h_1 will allow the receiver to recover two labels for an index on which $(\mathsf{left}, b_{00}, b_{01})$ and $(\mathsf{right}, b_{10}, b_{11})$ differ, destroying garbled-circuit security.

Fixing size blow-up via deferred encryption. We fix the above issue via deferred encryption techniques [DG17, BLSV18, GHMR18, ABD+21], allowing the sender to defer the HE encryptions of $(\mathsf{lb}_{i,b})$ labels to the receiver herself at decryption time! To enable this technique, the receiver further hashes (h_0, h_1) such that during decryption, the receiver, through the evaluation of a garbled circuit, will obtain an HE encryption of $(\mathsf{lb}_{i,b})$ labels with respect to $h_{x[\mathsf{root}]}$, where $(\mathsf{lb}_{i,b})$ and (h_0, h_1) are as above. To do this, we have to explain how the receiver further hashes down h_0 and h_1, and how she can later perform deferred encryption. First, the receiver R computes the hash value $\mathsf{hr} := \mathsf{Hash}(\mathsf{pp}, (h_0, h_1, \mathsf{root}))$, and sends hr to S. Next the sender $S(x)$ garbles $\mathsf{F}[x]$ to get $(\widetilde{\mathsf{F}}, \mathsf{lb}_{i,b})$ as above. Then, he forms a circuit $\mathsf{G}[x, (\mathsf{lb}_{i,b})]$ with x and $(\mathsf{lb}_{i,b})$ hardwired, which on an input (h'_0, h'_1, u) outputs $\mathsf{HEnc}(h'_{x[u]}, (\mathsf{lb}_{i,b}))$. The sender garbles $\mathsf{G}[x, (\mathsf{lb}_{i,b})]$ to get $(\widetilde{\mathsf{G}}, (\mathsf{lb}'_{i,b}))$. Before proceeding, let us consider R's perspective. If R is given $\widetilde{\mathsf{G}}$ and the labels $(\mathsf{lb}'_{i,z'[i]})$, where $z' := (h_0, h_1, \mathsf{root})$, she can evaluate $\widetilde{\mathsf{G}}$ on these labels, which will in turn release an HE encryption of labels $(\mathsf{lb}_{i,b})$ under $h_{x[\mathsf{root}]}$, as desired. To ensure R only gets the $(\mathsf{lb}'_{i,z'[i]})$ labels, S encrypts the $\{\mathsf{lb}'_{i,b}\}$ labels under hr, and sends the resulting HE ciphertext cth', as well as $\widetilde{\mathsf{F}}$ and $\widetilde{\mathsf{G}}$ to R. From cth' and $z' := (h_0, h_1, \mathsf{root})$, R can only recover the labels $(\mathsf{lb}'_{i,z'[i]})$, as desired.

How can the receiver decrypt? The receiver will evaluate $\widetilde{\mathsf{G}}$ on the decrypted $(\mathsf{lb}'_{i,z'[i]})$ labels, releasing an HE encryption cth of $(\mathsf{lb}_{i,b})$ labels under $h_{x[\mathsf{root}]}$. The receiver does not know whether cth is encrypted under

h_0 or h_1, so she tries to decrypt with respect to the pre-images of both hash values and checks which one (if any) is valid. However, this results in the following security issue: an HE scheme is not guaranteed to hide the underlying hash value with respect to which an HE ciphertext was made. For example, imagine an HE scheme where $\mathsf{HEnc}(h, (m_{i,b}))$ appends h to the ciphertext. Employing such an HE scheme (which is semantically secure) in the above construction will signal to the receiver if cth' was encrypted under h_0 or h_1, namely the bit value of $x[\mathsf{root}]$. This breaks sender security if $\mathsf{BP}(x) = 0$. Moreover, even if the HE encryption scheme is anonymous in the sense of hiding h, decrypting an h_b-formed HE encryption under the pre-image of h_{1-b} may result in $\perp$, or in junk labels that do not work on $\widetilde{\mathsf{F}}$. We use the same technique as in [ABD+21] of using anonymous hash encryption and garbled circuits to resolve this issue.

Signalling the correct output of F. In the above examples, $\mathsf{F}[x]$ outputs either accept or reject, indicating if $\mathsf{BP}(x)$ equals 1 or 0, respectively. But, in the desired functionality, $\mathsf{F}[x]$ outputs x if $\mathsf{BP}(x) = 1$. We cannot simply modify $\mathsf{F}[x]$ to output x if $q_{x_j} = \mathsf{accept}$ since in that case if the receiver evaluates $\widetilde{\mathsf{F}}$ on junk labels she will not be able to tell the difference between the junk output and x. Similar to [ABD+21], we address this problem by having S include a signal value in the ciphertext and in their message to R. Then we can modify $\mathsf{F}[x]$ to output x and the decrypted signal value. The receiver compares this output signal value with the true value. If they are equal, R knows that output x is not junk.

Handling unbalanced branching programs. The above discussion can be naturally extended to the balanced BP setting, wherein we have a full binary tree of depth d. When the BP is unbalanced, like our BPs for PSI and PSU, the above approach does not work, because the sender does not know *a priori* which branches stop prematurely. We solve this issue via the following technique. We design the circuit G to work in two modes: normal mode (as explained above) and halting mode, which is triggered when its input signals a leaf node. In halting mode, the circuit G will release its hardwired input x, assuming the halt is an accept. Executing the above blueprint requires striking a delicate balance to have both correctness and security.

Comparison with [ABD+21]. At a high level, the garbled-circuit-based laconic PSI construction of [ABD+21] is an ad hoc and specific instantiation of our general methodology. In particular, for a receiver with $m = 2^k$ elements (for $k := \mathsf{polylog}(\lambda)$), the construction of [ABD+21] builds a full binary tree of depth k, with the m elements appearing sorted in the leaves, Merkle hashed all the way up in a specific way. In particular, the pre-image of each node's hash value is comprised of its two children's hashes as well as some additional encoded information about its sub-tree, enabling an evaluator, with an input x, to make a deterministic left-or-right downward choice at each intermediate node. This is a very specific BP instantiation of PSI, where the intermediate BP nodes, instead of running index predicates (e.g., going left/right if the ith bit is 0/1), they run full-input predicates $\Phi : x \mapsto \{0, 1\}$, where Φ is defined based on the left sub-tree of the node. Our approach, on the other hand, handles branching programs for index predicates, and we subsume the results of [ABD+21] as a special case. In particular, we show how to design simple index-predicate BPs for PSI, PSU, and wildcard matching, the latter two problems are not considered by [ABD+21].

In summary, our construction generalizes and simplifies the approach of [ABD+21], getting much more mileage out of the garbled-circuit based approach. For example, [ABD+21] builds a secure protocol for a specific PSI-based BP which is in fact a decision tree: namely, the in-degree of all internal nodes is one. On the other hand, we generalize this concept to handle all decision trees and even the broader class of branching programs, in which the in-degree of intermediates nodes can be greater than one. Moreover, we introduce some new techniques (e.g., for handling unbalanced BPs) that may be of independent interest.

Comparison with [DGGM19]. The work of Döttling, Garg, Goyal, and Malavolta [DGGM19] builds laconic conditional disclosure of secrets (CDS) involving a sender holding an NP instance x and a message m, and a receiver holding x and a potential witness w for x. If $R(x, w) = 1$, where R is the corresponding relation R, the receiver learns m; else, the receiver learns no information about m. They show how to build two-round laconic CDS protocols from CDH/LWE with polylogarithmic communication and polylogarithmic sender computation. The CDS setting is incomparable to ours. The closest resemblance is to think of x, the BP input, as the NP instance, and of the BP as the NP witness w. But then under CDS, the input x is not kept hidden from the receiver. In particular, it is not even clear whether laconic CDS implies laconic PSI.

3 Preliminaries

Throughout this work, λ denotes the security parameter. $\mathsf{negl}(\lambda)$ denotes a negligible function in λ, that is, a function that vanishes faster than any inverse polynomial in λ.

For $n \in \mathbb{N}$, $[n]$ denotes the set $\{1, \ldots, n\}$. If $x \in \{0,1\}^n$ then the bits of x can be indexed as $x[i] := x_i$ for $i \in [n]$, where $x = x_1 \ldots x_n$ (note that indexing begins at 1, not 0). $x := y$ is used to denote the assignment of variable x to the value y. If $\mathcal{A}$ is a deterministic algorithm, $y \leftarrow \mathcal{A}(x)$ denotes the assignment of the output of $\mathcal{A}(x)$ to variable y. If $\mathcal{A}$ is randomized, $y \xleftarrow{\$} \mathcal{A}(x)$ is used. If S is a (finite) set, $x \xleftarrow{\$} S$ denotes the experiment of sampling uniformly at random an element x from S. If D is a distribution over S, $x \xleftarrow{\$} D$ denotes the element x sampled from S according to D. If D_0, D_1 are distributions, we say that D_0 is statistically (resp. computationally) indistinguishable from D_1, denoted by $D_0 \approx_s D_1$ (resp. $D_0 \stackrel{c}{\equiv} D_1$), if no unbounded (resp. PPT) adversary can distinguish between the distributions except with probability at most $\mathsf{negl}(\lambda)$.

If Π is a two-round two-party protocol, then $(\mathsf{m}_1, \mathsf{m}_2) \leftarrow \mathsf{tr}^{\Pi}(x_0, x_1, \lambda)$ denotes the protocol transcript, where x_i is party P_i's input for $i \in \{0,1\}$. For $i \in \{0,1\}$, $(x_i, r_i, \mathsf{m}_1, \mathsf{m}_2) \leftarrow \mathsf{view}_i^{\Pi}(x_0, x_1, \lambda)$ denotes P_i's "view" of the execution of Π, consisting of their input, random coins, and the protocol transcript.

Definition 1 (Computational Diffie-Hellman) *Let $\mathcal{G}(\lambda)$ be an algorithm that outputs $(\mathbb{G}, p, g)$ where $\mathbb{G}$ is a group of prime order p and g is a generator of the group. The CDH assumption holds for generator $\mathcal{G}$ if for all PPT adversaries $\mathcal{A}$*

$$\Pr\left[g^{a_1 a_2} \leftarrow \mathcal{A}(\mathbb{G}, p, g, g^{a_1}, g^{a_2}) : \begin{array}{c}(\mathbb{G}, p, g) \leftarrow \mathcal{G}(\lambda) \\ a_1, a_2 \leftarrow\!\!\$\ \mathbb{Z}_p\end{array}\right] \leq \mathsf{negl}(\lambda).$$

Definition 2 (Learning with Errors) *Let $q, k \in \mathbb{N}$ where $k \in \mathsf{poly}(\lambda)$, $\mathbf{A} \in \mathbb{Z}_q^{k \times n}$ and $\beta \in \mathbb{R}$. For any $n = \mathsf{poly}(k \log q)$, the LWE assumption holds if for every PPT algorithm $\mathcal{A}$ we have*

$$|\Pr[1 \leftarrow \mathcal{A}(\mathbf{A}, \mathbf{sA} + \mathbf{e})] - \Pr[1 \leftarrow \mathcal{A}(\mathbf{A}, \mathbf{y})]| \leq \mathsf{negl}(\lambda)$$

for $\mathbf{s} \xleftarrow{\$} \{0,1\}^k$, $\mathbf{e} \xleftarrow{\$} D_{\mathbb{Z}^n, \beta}$ and $\mathbf{y} \xleftarrow{\$} \{0,1\}^n$, where $D_{\mathbb{Z}^n, \beta}$ is some error distribution.

The following definitions related to branching programs are modified from [IP07].

Definition 3 (Branching Program (BP)) *A (deterministic) branching program over the input domain $\{0,1\}^\lambda$ and output domain $\{0,1\}$ is defined by a tuple (V, E, T, Val) where:*

- *$G := (V, E)$ is a directed acyclic graph of depth d.*
- *Two types of nodes partition V:*
  - *Interior nodes: Have outdegree 2.[7] The root node, denoted $v_1^{(0)}$, has indegree 0.*
  - *Terminal/leaf nodes: Have outdegree 0. T denotes the set of terminal nodes. Leaf nodes are labeled as $T = \{u_1, \ldots, u_{|T|}\}$. Each $u_i \in T$ encodes a value in $\{0,1\}$.*
- *For every non-root node $u \in V \setminus \{v_1^{(0)}\}$ there exists a path from $v_1^{(0)}$ to u.*
- *Each node in V encodes a value in $[\lambda]$. These values are stored in the array Val such that for all $v \in V$, $\mathsf{Val}[v] = i$ for some $i \in [\lambda]$.*
- *The elements of the edge set E are formatted as an ordered tuple (v, v', b) indicating a directed edge from $v \in V$ to $v' \in V$ with label $b \in \{0,1\}$. If $b = 0$ (resp. $b = 1$), v' is the left (resp. right) child of v.*

BP Evaluation. The output of a branching program is defined by the function $\mathsf{BP} : \{0,1\}^\lambda \to \{0,1\}$, which on input $x \in \{0,1\}^\lambda$ outputs a bit. Evaluation of BP (see Fig. 2, right, and relevant function definitions below) follows the unique path in G induced by x from the root $v_1^{(0)}$ to a leaf node $u \in T$. The output of BP is the value encoded in u, $\mathsf{Val}[u]$.

- $\Gamma : V \setminus T \times \{0,1\} \to V$ takes as input an internal node v and a bit b and outputs v's left child if $b = 0$ and v's right child if $b = 1$.
- $\mathsf{Eval}_{\mathsf{int}} : V \setminus T \times \{0,1\}^\lambda \to V$ takes as input an interior node v and a string of length λ and outputs either v's left or right child ($\Gamma(v, 0)$ or $\Gamma(v, 1)$, respectively). See Figure 2, left.
- $\mathsf{Eval}_{\mathsf{leaf}} : T \to \{0,1\}$ takes as input a terminal node $u \in T$ and outputs the value $\mathsf{Val}[u]$.

$\mathsf{Eval}_{\mathsf{int}}(v, x)$:	$\mathsf{BP}(x)$:
$i \leftarrow \mathsf{Val}[v]$	$v \leftarrow v_1^{(0)}$
If $x[i] = 0$ then return $\Gamma(v, 0)$	While $v \notin T$ do $v \leftarrow \mathsf{Eval}_{\mathsf{int}}(v, x)$
Else return $\Gamma(v, 1)$	$y \leftarrow \mathsf{Eval}_{\mathsf{leaf}}(v)$
	Return y

Fig. 2: Interior node evaluation function $\mathsf{Eval}_{\mathsf{int}}$ and BP evaluation function BP.

Definition 4 (Layered BP) *A BP of depth d is layered if the node set V can be partitioned into $d+1$ disjoint levels $V = \bigcup_{i=0}^{d} V^{(i)}$, such that $V^{(0)} = \{v_1^{(0)}\}$, $V^{(d)} = T$, and for every edge $e = (u, v)$ we have $u \in V^{(i)}$, $v \in V^{(i+1)}$ for some level $i \in \{0, \ldots, d\}$. We refer to $V^{(i)}$ as the i-th level of the BP, or the level at depth i. Nodes on level i are labelled from leftmost to rightmost: $V^{(i)} = \{v_1^{(i)}, \ldots, v_{|V^{(i)}|}^{(i)}\}$.*

We require that all branching programs in this work are layered.

4 Semi-Honest Laconic 2PC with Branching Programs

Our construction uses hash encryption schemes with garbled circuits. The following definitions are taken directly from [ABD+21].

Definition 5 (Hash Encryption [DG17, BLSV18]) *A hash encryption scheme* $\mathsf{HE} = (\mathsf{HGen}, \mathsf{Hash}, \mathsf{HEnc}, \mathsf{HDec})$ *and associated security notions are defined as follows.*

- $\mathsf{HGen}(1^\lambda, n)$*: Takes as input a security parameter 1^λ and an input size n and outputs a hash key* pp.
- $\mathsf{Hash}(\mathsf{pp}, z)$*: Takes as input a hash key* pp *and $z \in \{0,1\}^n$, and deterministically outputs $h \in \{0,1\}^\lambda$.*
- $\mathsf{HEnc}(\mathsf{pp}, h, \{m_{i,b}\}_{i\in[n],b\in\{0,1\}}; \{r_{i,b}\})$*: Takes as input a hash key* pp*, a hash output h, messages $\{m_{i,b}\}$ and randomness $\{r_{i,b}\}$, and outputs $\{\mathsf{cth}_{i,b}\}_{i\in[n],b\in\{0,1\}}$, (written concisely as $\{\mathsf{cth}_{i,b}\}$). Each ciphertext $\mathsf{cth}_{i,b}$ is computed as $\mathsf{cth}_{i,b} = \mathsf{HEnc}(\mathsf{pp}, h, m_{i,b}, (i, b); r_{i,b})$, where we have overloaded the* HEnc *notation.*
- $\mathsf{HDec}(z, \{\mathsf{cth}_{i,b}\})$*: Takes as input a hash input z and $\{\mathsf{cth}_{i,b}\}$ and outputs n messages $(m_1, \ldots, m_n)$. Correctness is required such that for the variables above, $(m_1, \ldots, m_n) = (m_{1,z[1]}, \ldots, m_{n,z[n]})$.*
- ***Semantic Security****: Given $z \in \{0,1\}^n$, no adversary can distinguish between encryptions of messages made to indices $(i, \bar{z}[i])$. For any PPT $\mathcal{A}$, sampling $\mathsf{pp} \xleftarrow{\$} \mathsf{HGen}(1^\lambda, n)$, if $(z, \{m_{i,b}\}, \{m'_{i,b}\}) \xleftarrow{\$} \mathcal{A}(\mathsf{pp})$ and if $m_{i,z[i]} = m'_{i,z[i]}$ for all $i \in [n]$, then $\mathcal{A}$ cannot distinguish between $\mathsf{HEnc}(\mathsf{pp}, h, \{m_{i,b}\})$ and $\mathsf{HEnc}(\mathsf{pp}, h, \{m'_{i,b}\})$, where $h \leftarrow \mathsf{Hash}(\mathsf{pp}, z)$.*
- ***Anonymous Semantic Security****: For a random $\{m_{i,b}\}$ with equal rows (i.e., $m_{i,0} = m_{i,1}$), the output of $\mathsf{HEnc}(\mathsf{pp}, h, \{m_{i,b}\})$ is pseudorandom even in the presence of the hash input. Formally, for any $z \in \{0,1\}^n$, sampling $\mathsf{pp} \xleftarrow{\$} \mathsf{HGen}(1^\lambda, n)$, $h \leftarrow \mathsf{Hash}(\mathsf{pp}, z)$, and sampling $\{m_{i,b}\}$ uniformly at random with the same rows, then $\boldsymbol{v} := (\mathsf{pp}, z, \mathsf{HEnc}(\mathsf{pp}, h, \{m_{i,b}\}))$ is indistinguishable from another tuple in which we replace the hash-encryption component of $\boldsymbol{v}$ with a random string.*

The following results are from [BLSV18, GGH19].

Lemma 1 *Assuming CDH/LWE there exists anonymous hash encryption schemes, where $n = 3\lambda$ (i.e., $\mathsf{Hash}(\mathsf{pp}, \cdot) \colon \{0,1\}^{3\lambda} \mapsto \{0,1\}^\lambda$).*[8] *Moreover, the hash function* Hash *satisfies robustness in the following sense: for any input distribution on z which samples at least 2λ bits of z uniformly at random, $(\mathsf{pp}, \mathsf{Hash}(\mathsf{pp}, z))$ and (pp, u) are statistically close, where $\mathsf{pp} \xleftarrow{\$} \mathsf{HGen}(1^\lambda, 3\lambda)$ and $u \xleftarrow{\$} \{0,1\}^\lambda$.*

We also review garbled circuits and the anonymous property, as defined in [BLSV18].

[7] We assume no nodes have outdegree 1 since such nodes can be removed from the BP w.l.o.g.

[8] The CDH construction of [BLSV18] satisfies a weaker notion of anonymity, in which only some part of the ciphertext is pseudorandom. But for ease of presentation, we keep the notion as is.

Definition 6 (Garbled Circuits) *A garbling scheme for a class of circuits* $\mathcal{C} := \{\mathsf{C}\colon \{0,1\}^n \mapsto \{0,1\}^m\}$ *consists of* $(\mathsf{Garb}, \mathsf{Eval}, \mathsf{Sim})$ *satisfying the following.*

- ***Correctness****: For all* $\mathsf{C} \in \mathcal{C}$, $\mathsf{m} \in \{0,1\}^n$, $\Pr[\mathsf{Eval}(\widetilde{\mathsf{C}}, \{\mathsf{lb}_{i,\mathsf{m}[i]}\}) = \mathsf{C}(\mathsf{m})] = 1$, *where* $(\widetilde{\mathsf{C}}, \{\mathsf{lb}_{i,b}\}) \xleftarrow{\$} \mathsf{Garb}(1^\lambda, \mathsf{C})$.
- ***Simulation Security****: For any* $\mathsf{C} \in \mathcal{C}$ *and* $\mathsf{m} \in \{0,1\}^n$: $(\widetilde{\mathsf{C}}, \{\mathsf{lb}_{i,\mathsf{m}[i]}\}) \stackrel{c}{\equiv} \mathsf{Sim}(1^\lambda, \mathsf{C}, \mathsf{C}(\mathsf{m}))$, *where* $(\widetilde{\mathsf{C}}, \{\mathsf{lb}_{i,b}\}) \xleftarrow{\$} \mathsf{Garb}(1^\lambda, \mathsf{C})$.
- ***Anonymous Security***[9] *[BLSV18]: For any* $\mathsf{C} \in \mathcal{C}$, *if the output of* $\mathsf{C}(x)$ *for* $x \in \{0,1\}^n$ *is uniformly random, then the output of* $\mathsf{Sim}(1^\lambda, \mathsf{C}, y)$ *is pseudorandom.*

Lemma 2 ([BLSV18]) *Anonymous garbled circuits can be built from one-way functions.*

Hash Encryption Notation. We assume $\mathsf{Hash}(\mathsf{pp}, \cdot)\colon \{0,1\}^n \mapsto \{0,1\}^\lambda$, where $n = 3\lambda$. $\{\mathsf{lb}_{i,b}\}$ denotes a sequence of pairs of labels, where $i \in [n]$ and $b \in \{0,1\}$. For $\boldsymbol{r} := \{r_{i,b}\}$, $\mathsf{HEnc}(\mathsf{pp}, h, \{\mathsf{lb}_{i,b}\}; \boldsymbol{r})$ denotes ciphertexts $\{\mathsf{cth}_{i,b}\}$, where $\mathsf{cth}_{i,b} = \mathsf{HEnc}(\mathsf{pp}, h, \mathsf{lb}_{i,b}, (i,b); r_{i,b})$. We overload notation as follows. $\{\mathsf{lb}_i\}$ denotes a sequence of 3λ elements. For $\boldsymbol{r} := \{r_{i,b}\}$, $\mathsf{HEnc}(\mathsf{pp}, h, \{\mathsf{lb}_i\}; \boldsymbol{r})$ denotes a hash encryption where both plaintext rows are $\{\mathsf{lb}_i\}$; namely, ciphertexts $\{\mathsf{cth}_{i,b}\}$, where $\mathsf{cth}_{i,b} = \mathsf{HEnc}(\mathsf{pp}, h, \{m_{i,b}\}; r_{i,b})$ and $m_{i,0} = m_{i,1} = \mathsf{lb}_i$, for all i.

Definition 7 (BP-2PC Functionality) *We define the evaluation of a branching program in the two-party communication setting (*BP-2PC*) to be a two-round protocol between a receiver* R *and a sender* S *such that:*

- R *holds a branching program* BP *with evaluation function* $\mathsf{BP} : \{0,1\}^\lambda \to \{0,1\}$ *and* S *holds a string* $\mathsf{id} \in \{0,1\}^\lambda$. *In the first round of the protocol,* R *sends the message* m_1 *to* S. *In the second round* S *sends* m_2 *to* R.
- ***Correctness:*** *If* $\mathsf{BP}(\mathsf{id}) = 1$, *then* R *outputs* id. *Otherwise,* R *outputs* $\perp$.
- ***Computational (resp., statistical) receiver security:*** BP-2PC *achieves receiver security if for all* $\mathsf{id} \in \{0,1\}^\lambda$, *and all pairs of branching programs* $\mathrm{BP}_0, \mathrm{BP}_1$ *we have that* $\mathsf{view}_S^{\mathsf{BP\text{-}2PC}}(\mathrm{BP}_0, \mathsf{id}, \lambda) \approx \mathsf{view}_S^{\mathsf{BP\text{-}2PC}}(\mathrm{BP}_1, \mathsf{id}, \lambda)$. *If the distributions are computationally (resp., statistically) indistinguishable then we have computational (resp., statistical) security.*
- ***Computational (resp., statistical) sender security:*** BP-2PC *achieves sender security if for all branching programs* BP, *and all pairs* $\mathsf{id}_0, \mathsf{id}_1 \in \{0,1\}^\lambda$ *with* $\mathsf{BP}(\mathsf{id}_0) = 0 = \mathsf{BP}(\mathsf{id}_1)$, *we have that* $\mathsf{view}_R^{\mathsf{BP\text{-}2PC}}(\mathrm{BP}, \mathsf{id}_0, \lambda) \approx \mathsf{view}_R^{\mathsf{BP\text{-}2PC}}(\mathrm{BP}, \mathsf{id}_1, \lambda)$. *If the distributions are computationally (resp. statistically) indistinguishable, we have computational (resp. statistical) security.*

4.1 The BP-2PC Construction

In this section, we give a construction for a BP-2PC protocol, inspired by laconic OT techniques [CDG+17, ABD+21]. Construction 1 uses hash encryption and garbling schemes. A high-level overview is as follows.

1. The receiver party R hashes their branching program in a 'specific way' from the leaf level up to the root. R then sends the message $\mathsf{m}_1 = (\mathsf{d}_\mathsf{m}, \mathsf{h}_\mathsf{root})$ to the sender, where d_m is the maximum BP depth and h_root is the hash value of the root node of the hashed BP.
2. The sender party S gets the message $\mathsf{m}_1 = (\mathsf{d}_\mathsf{m}, \mathsf{h}_\mathsf{root})$ and garbles one circuit for every possible level of the hash tree, (i.e., generates d_m garbled circuits). S starts with the leaf level and garbles circuit F (Fig. 3). F takes as input a leaf node value and two random strings. If the leaf node value is 1, F outputs the hardcoded sender element id and a random, fixed, signal string r. Otherwise, F outputs two random strings (id', r'). Then for every level from the leaf parents to the root, S garbles the circuit V (also in Fig. 3). Each V garbled by the sender has the labels of the previously generated garbled circuit hardcoded. After garbling, S computes a hash encryption of the root-level garbled circuit labels using the hash image h_root. Finally, S sends $\mathsf{m}_2 := (\widetilde{\mathsf{C}}_{\mathsf{d}_\mathsf{m}}, \ldots, \widetilde{\mathsf{C}}_0, \{\mathsf{cth}_{i,b}^{(0)}\}, r)$ to R, where $\widetilde{\mathsf{C}}_w$ is the garbled circuit associated with level w, $\{\mathsf{cth}_{i,b}^{(0)}\}$ is the encryption of the labels for $\widetilde{\mathsf{C}}_0$, and r is the signal value.
3. For all root-to-leaf paths through the BP, R runs DecPath (Fig. 3, bottom) on m_2 searching for the path that will decrypt to a signal value equal to r from m_2. On input a path pth and m_2, DecPath outputs a pair $(\mathsf{id}_\mathsf{pth}, r_\mathsf{pth})$ to R. If $r_\mathsf{pth} = r$, then R takes id_pth to be S's element.

[9] called *blind* garbled circuits in [BLSV18].

Construction 1 (BP-2PC) *We require the following ingredients for the two-round, two-party communication BP construction.*

1. *An anonymous and robust hash encryption scheme* $\mathsf{HE} = (\mathsf{HGen}, \mathsf{Hash}, \mathsf{HEnc}, \mathsf{HDec})$*, where* $\mathsf{Hash}(\mathsf{pp}, \cdot): \{0,1\}^{3\lambda} \mapsto \{0,1\}^{\lambda}$.
2. *An anonymous garbling scheme* $\mathsf{GS} = (\mathsf{Garb}, \mathsf{Eval}, \mathsf{Sim})$.
3. *Circuits* F, V, *and procedure* $\mathsf{DecPath}$, *defined in Figure 3.*

The receiver holds a—potentially unbalanced—branching program BP of depth $d \leq \lambda+1$ *as defined in Def. 3. The sender has a single element* $\mathsf{id} \in \{0,1\}^{\lambda}$. $\mathsf{BP\text{-}2PC} := (\mathsf{GenCRS}, \mathsf{R}_1, \mathsf{S}, \mathsf{R}_2)$ *is a triple of algorithms built as follows.*

$\mathsf{GenCRS}(1^{\lambda})$*: Return* $\mathsf{crs} \xleftarrow{\$} \mathsf{HGen}(1^{\lambda}, 3\lambda)$.

$\mathsf{R}_1(\mathsf{crs}, \mathsf{BP})$*: BP has terminal node set* $T = \{u_1, \ldots, u_{|T|}\}$*. Nodes in level* $0 \leq w \leq d$ *are labelled from leftmost to rightmost:* $V^{(w)} = \{v_1^{(w)}, \ldots, v_{|V^{(w)}|}^{(w)}\}$.

- *Parse* $\mathsf{crs} := \mathsf{pp}$*. The receiver creates a hashed version of BP, beginning at the leaf level: For* $j \in [|T|]$*, sample* $x_j, x_j' \xleftarrow{\$} \{0,1\}^{\lambda}$ *and compute* $h_j^{(d)} \leftarrow \mathsf{Hash}(\mathsf{pp}, (\mathsf{Val}[u_j]^{\times\lambda}, x_j, x_j'))$. $\mathsf{Val}[u_j]^{\times\lambda}$ *indicates that* $\mathsf{Val}[u_j]$ *is copied* λ *times to obtain either the all zeros or all ones string of length* λ.
  The remaining levels are hashed from level $d-1$ *up to* 0 *(the root):*
  1. *For* w *from* $d-1$ *to* 0, $|V^{(w)}|$ *nodes are added to level* w*. The hash value of each node is the hash of the concatenation of its left child, right child, and the index encoded in the current node. Formally: For* $j \in [|V^{(w)}|]$*, set* $h_j^{(w)} \leftarrow \mathsf{Hash}(\mathsf{pp}, (h_{2j-1}^{(w+1)}, h_{2j}^{(w+1)}, \mathsf{Val}[v_j^{(w)}]))$*, where* $\mathsf{Val}[v_j^{(w)}]$ *is the value of the bit encoded in the* j*-th node of level* w*. If needed, padding is added so that* $|\mathsf{Val}[v_j^{(w)}]| = \lambda$.
  2. *Let* $\mathsf{m}_1 := (\mathsf{d_m}, \mathsf{h_{root}})$*, where* $\mathsf{d_m} = \lambda + 1$ *is the maximum tree depth and* $\mathsf{h_{root}} := h_1^{(0)}$ *is the root hash value. For all* $i \in [|T|]$*,* $w \in \{0, \ldots, d\}$*, and* $j \in [|V^{(w)}|]$*, set* $\mathsf{st} := (\{x_i\}, \{x_i'\}, \{v_j^{(w)}\})$*. Send* m_1 *to* S.

$\mathsf{S}(\mathsf{crs}, \mathsf{id}, \mathsf{m}_1)$*:*

- *Parse* $\mathsf{m}_1 := (\mathsf{d_m}, \mathsf{h_{root}})$ *and* $\mathsf{crs} := \mathsf{pp}$*. Sample* $r, \mathsf{id}', r' \xleftarrow{\$} \{0,1\}^{\lambda}$ *and padding* $\mathsf{pad} \xleftarrow{\$} \{0,1\}^{2(n-1)}$*. Let* $\mathsf{C_{d_m}} := \mathsf{F}[\mathsf{id}, \mathsf{id}', r, r']$ *(Fig. 3). Garble* $(\widetilde{\mathsf{C}}_{\mathsf{d_m}}, \{\mathsf{lb}_{i,b}^{(\mathsf{d_m})}\}) \xleftarrow{\$} \mathsf{Garb}(\mathsf{C_{d_m}})$*. For* w *from* $\mathsf{d_m} - 1$ *to* 0 *do:*
  1. *Sample random* $\boldsymbol{r}_w$ *and let* $\mathsf{C}_w := \mathsf{V}[\mathsf{pp}, \mathsf{id}, \{\mathsf{lb}_{i,b}^{(w+1)}\}, \boldsymbol{r}_w, r, \mathsf{id}', r', \mathsf{pad}]$.
  2. *Garble* $(\widetilde{\mathsf{C}}_w, \{\mathsf{lb}_{i,b}^{(w)}\}) \xleftarrow{\$} \mathsf{Garb}(\mathsf{C}_w)$.
- *Let* $\{\mathsf{cth}_{i,b}^{(0)}\} \xleftarrow{\$} \mathsf{HEnc}(\mathsf{pp}, \mathsf{h_{root}}, \{\mathsf{lb}_{i,b}^{(0)}\})$.
- *Send* $\mathsf{m}_2 := (\widetilde{\mathsf{C}}_{\mathsf{d_m}}, \ldots, \widetilde{\mathsf{C}}_0, \{\mathsf{cth}_{i,b}^{(0)}\}, r)$ *to* R_2.

$\mathsf{R}_2(\mathsf{crs}, \mathsf{st}, \mathsf{m}_2)$*: Parse* $\mathsf{st} := (\{x_i\}, \{x_i'\}, \{v_j^{(w)}\})$ *and* $\mathsf{m}_2 := (\widetilde{\mathsf{C}}_{\mathsf{d_m}}, \ldots, \widetilde{\mathsf{C}}_0, \{\mathsf{cth}_{i,b}^{(0)}\}, r)$*.* $\forall$ *leaves* $u \in T$*, let* $\mathsf{pth}_u := ((\mathsf{Val}[u]^{\times\lambda}, x, x'), \ldots, \mathsf{h_{root}})$ *be the root to leaf* u *path in BP. Let* ℓ *be the length of* pth_u *and let*

$$(\mathsf{id}_u, r_u) \leftarrow \mathsf{DecPath}(\mathsf{pth}_u, \widetilde{\mathsf{C}}_{\mathsf{d_m}}, \ldots, \widetilde{\mathsf{C}}_0, \{\mathsf{cth}_{i,b}^{(0)}\}).$$

If $r_u = r$*, then output* id_u *and halt. If for all* $u \in T$*,* $r_u \neq r$*, then output* $\perp$.

R_2 must run $\mathsf{DecPath}$ on every root-to-leaf path. R_2 is written above as if there is a unique path from the root to each leaf. But since we allow nodes to have in-degree > 1, a leaf may be reachable from more than one path. In such a case, the path iteration in R_2 should be modified so that all paths are explored.

Communication costs. The first message m_1 is output by R_1 and sent to S. m_1 consists of the maximum depth $\mathsf{d_m}$ and the hash digest $\mathsf{h_{root}}$, which are $O(\log \lambda)$ and λ bits, respectively. So the receiver's communication cost is $\mathsf{poly}(\mathsf{d_m}, \lambda)$, and since we assume $\mathsf{d_m} = \lambda + 1$, this is $\mathsf{poly}(\mathsf{d_m}, \lambda)$. Next, m_2 is output by S and sent to R_2. m_2 consists of the following:

- $\widetilde{\mathsf{C}}_0$: the garbling of circuit F. F has 4λ bits hardcoded (including id).

Circuit $\mathsf{F}[\mathsf{id}, \mathsf{id}', r, r'](v, x, x')$**:**
HARDWIRED: Target identity id, signal value r, and random strings id', r'.
OPERATION: Return

$$\begin{cases} \{\mathsf{id}, r\} & \text{if } v = 1^\lambda \\ \{\mathsf{id}', r'\} & \text{otherwise} \end{cases}$$

Circuit $\mathsf{V}[\mathsf{pp}, \mathsf{id}, \{\mathsf{lb}_{i,b}\}, \boldsymbol{r}, r, \mathsf{id}', r', \mathsf{pad}](a, b, c)$**:**
HARDWIRED: Hash public parameter pp, target identity id, labels $\{\mathsf{lb}_{i,b}\}$, HEnc randomness $\boldsymbol{r}$, signal value r, random strings id', r', and padding pad.
OPERATION:
If $a = 1^\lambda$ then return $\{\mathsf{id}, r, \mathsf{pad}\}$
If $a = 0^\lambda$ then return $\{\mathsf{id}', r', \mathsf{pad}\}$
Else set $h_1 \leftarrow a$, $h_2 \leftarrow b$, $i \leftarrow c$ and return

$$\{\mathsf{cth}_{i,b}\} \leftarrow \begin{cases} \mathsf{HEnc}(\mathsf{pp}, h_1, \{\mathsf{lb}_{i,b}\}; \boldsymbol{r}) & \text{if } \mathsf{id}[i] = 0 \\ \mathsf{HEnc}(\mathsf{pp}, h_2, \{\mathsf{lb}_{i,b}\}; \boldsymbol{r}) & \text{otherwise} \end{cases}$$

Procedure $\mathsf{DecPath}(\mathsf{pth}, \mathsf{m}_2)$**:**
INPUT: A leaf-root path pth of length $\ell \le d$ and tuple $\mathsf{m}_2 := (\widetilde{\mathsf{C}}_{\mathsf{d_m}}, \ldots, \widetilde{\mathsf{C}}_0, \{\mathsf{cth}^{(0)}_{i,b}\}, r)$.
OPERATION: 1. Parse

$$\mathsf{pth} := ((\underbrace{\mathsf{Val}[v^{(\ell)}]^{\times\lambda}, x, x'}_{z_\ell}), \underbrace{(h^{(\ell)}, h'^{(\ell)}, \mathsf{Val}[v^{(\ell-1)}])}_{z_{\ell-1}}, \ldots, \underbrace{(h^{(1)}, h'^{(1)}, \mathsf{Val}[v^{(0)}])}_{z_0}, \mathsf{h}_{\mathsf{root}}) .$$

2. For w from 0 to $\ell - 1$ do:
   (a) Let $\{\mathsf{lb}^{(w)}_i\} \leftarrow \mathsf{HDec}(z_w, \{\mathsf{cth}^{(w)}_{i,b}\})$. (b) Set $\{\mathsf{cth}^{(w+1)}_{i,b}\} \leftarrow \mathsf{Eval}(\widetilde{\mathsf{C}}_w, \{\mathsf{lb}^{(w)}_i\})$.
3. Let $\{\mathsf{lb}^{(\ell)}_i\} \leftarrow \mathsf{HDec}(z_\ell, \{\mathsf{cth}^{(\ell)}_{i,b}\})$.
4. Let $\{\mathsf{id}_{\mathsf{pth}}, r_{\mathsf{pth}}, \mathsf{pad}\} \leftarrow \mathsf{Eval}(\widetilde{\mathsf{C}}_\ell, \{\mathsf{lb}^{(\ell)}_i\})$ and return $(\mathsf{id}_{\mathsf{pth}}, r_{\mathsf{pth}})$.

Fig. 3: Circuits F, V and procedure $\mathsf{DecPath}$ for construction 1. See Fig. 9 for an illustration of $\mathsf{DecPath}$. Circuits based on those in Table 1 of [ABD+21].

- $\widetilde{\mathsf{C}}_i$ for $i \in [\mathsf{d_m}]$: each $\widetilde{\mathsf{C}}_i$ is a garbling of circuit V. Hardcoded in each V is the public parameter, $2n = 6\lambda$ garbled circuit labels, hash encryption randomness, and an additional $\mathsf{poly}(\lambda)$ bits (including id).
- $\{\mathsf{cth}^{(0)}_{i,b}\}_{i\in[n],b\in\{0,1\}}$: 6λ hash encryption ciphertexts. Each cth is the hash encryption of one garbled circuit label.
- r: the λ-bit signal string.

So the sender's communication cost grows with $\mathsf{poly}(\lambda, \mathsf{d_m}, |\mathsf{id}|)$, which is $\mathsf{poly}(\lambda)$.

Computation costs.

R_1: performs $|V|$ Hash evaluations and samples $2|T|$ random strings of length λ.
S: samples $\mathsf{poly}(\lambda, \mathsf{d_m})$ random bits, garbles an F circuit, garbles $\mathsf{d_m}$ V circuits, and performs a hash encryption of 6λ garbled labels. The sender's computation cost does not depend on the total size of the BP, just $\mathsf{d_m}$.
R_2: runs $\mathsf{DecPath}$ for every root-leaf path. Each iteration of $\mathsf{DecPath}$ requires at most $\mathsf{d_m} + 1$ HDec and garbled circuit Eval evaluations (all, but possibly one, Eval will be w.r.t. a V circuit).

In total, R's computation cost is $O(\lambda, \mathsf{d_m}, |V|, |\mathsf{PTH}|)$, where $|\mathsf{PTH}|$ is the total number of root-to-leaf paths in the BP. So we require that $\mathsf{d_m} = \lambda + 1$ and $|V|$ and $|\mathsf{PTH}|$ are $\mathsf{poly}(\lambda)$. The sender's computation cost is $\mathsf{poly}(\lambda, \mathsf{d_m})$.

Lemma 3 *Construction 1 is correct in the sense that (1) if* $\mathsf{BP}(\mathsf{id}) = 1$*, then with overwhelming probability* R_2 *outputs* id *and (2) if* $\mathsf{BP}(\mathsf{id}) = 0$*, then with overwhelming probability* R_2 *outputs* $\perp$.

Theorem 1. *If* HE *is an anonymous and robust hash encryption (defined in Lemma 1), and* GS *is an anonymous garbling scheme, then the* BP-2PC *protocol of Construction 1 provides statistical security for the receiver and semi-honest security for the sender.*

The proofs of Lemma 3 and Theorem 1 are in Sections 6 and 7, respectively.

5 Applications

Construction 1 can be used to realize multiple functionalities by reducing the desired functionality to an instance of BP-2PC. One step of the reductions involves constructing a branching program based on a set of bit strings.

At a high level, SetBP (Fig. 4) creates a branching program for a set of elements $S := \{x_1, \ldots, x_m\}$ in three main steps. For concreteness, suppose the goal is to use this BP for a private set intersection.

First, for every prefix $a \in \{\epsilon\} \cup \{0,1\} \cup \{0,1\}^2 \cup \cdots \cup \{0,1\}^\lambda$ of the elements in S, a node v_a is added to the set of nodes V. If $a \in S$, then the value encoded in v_a is set to 1; this is an 'accept' leaf. If $|a| < \lambda$, then the encoded value is set to $|a|+1$. When the BP is being evaluated on some input, $|a|+1$ will indicate the bit following prefix a. Next, edges are created between the BP levels. For $|a| < \lambda$, if for $b \in \{0,1\}$, node $v_{a\|b}$ exists in V, then a b-labelled edge is added from v_a to $v_{a\|b}$. For $b \in \{0,1\}$, if $v_{a\|b} \notin V$, then node $v_{a\|b}$ is added to V with an encoded bit 0. This is a 'reject' leaf. Then a b-labelled edge is added from v_a to $v_{a\|b}$. Finally, the BP is pruned. If two sibling leaves are both encoded with the same value, they are deleted and their parent becomes a leaf encoding that same value.

The definition below generalizes this concept by allowing us to capture both PSI and PSI via an indicator bit b_{pth}. In the description above, b_{pth} is set to 1 for the PSI setting. For PSU we set $b_{\mathsf{pth}} = 0$.

Construction 2 (Set to branching program) *Figure 4 defines a procedure to create a branching program from an input set S. $\mathsf{SetBP}(S, b_{\mathsf{pth}})$ takes as input a set $S := \{x_1, \ldots, x_m\}$ of m strings, all of length λ and a bit b_{pth} and outputs a tuple (V, E, T, Val) defining a branching program. The output BP is such that if $x \in S$, then $\mathsf{BP}(x) = b_{\mathsf{pth}}$, and if $x \notin S$, then $\mathsf{BP}(x) = 1 - b_{\mathsf{pth}}$.*

Procedure SetBP *runs in time $O(\lambda|S|)$. In particular, when $|S| = \mathsf{poly}(\lambda)$,* SetBP *generates the BP in time $O(\mathsf{poly}(\lambda))$. The output BP has depth $d \le \lambda + 1$ and the number of nodes is $2d + 1 \le |V| \le 2^{d+1} - 1$. Evaluation of $\mathsf{BP}(x)$ for arbitrary $x \in \{0,1\}^\lambda$ takes time $O(\mathsf{poly}(\lambda))$.*

BP evaluation runtime: Recall the BP evaluation algorithm in Fig. 2. Each loop iteration of the evaluation makes progress by moving down the tree one level. The number of iterations is at most the tree depth, which is at most $\lambda + 1$ for the BP created in Fig. 4. Each iteration takes constant time, so evaluation of $\mathsf{BP}(x)$ for arbitrary $x \in \{0,1\}^\lambda$ takes time $O(\mathsf{poly}(\lambda))$.

5.1 Private Set Intersection (PSI)

Assume a sender party has a set $S_{\mathsf{S}} = \{\mathsf{id}\}$ where $\mathsf{id} \in \{0,1\}^\lambda$ and a receiver has a polynomial-sized set $S_{\mathsf{R}} \subset \{0,1\}^\lambda$. In this setting, we define PSI as follows.

Definition 8 (Private set Intersection (PSI) functionality with $|S_{\mathsf{S}}| = 1$) *Let Π be a two-party communication protocol. Let R be the receiver holding set $S_{\mathsf{R}} \subset \{0,1\}^\lambda$ and let S be the sender holding singleton set $S_{\mathsf{S}} = \{\mathsf{id}\}$, with $\mathsf{id} \in \{0,1\}^\lambda$. Π is a PSI protocol if the following hold after it is executed.*

- ***Correctness:*** *R learns $S_{\mathsf{R}} \cap \{\mathsf{id}\}$ if and only if $\mathsf{id} \in S_{\mathsf{R}}$.*
- ***Receiver security:*** *Π achieves receiver security if $\forall \mathsf{id} \in \{0,1\}^\lambda$, and all pairs $S_{\mathsf{R}0}, S_{\mathsf{R}1} \subset \{0,1\}^\lambda$ we have that $\mathsf{view}_S^\Pi(S_{\mathsf{R}0}, \mathsf{id}, \lambda) \approx \mathsf{view}_S^\Pi(S_{\mathsf{R}1}, \mathsf{id}, \lambda)$. If the distributions are computationally (resp., statistically) indistinguishable then we have computational (resp., statistical) security.*
- ***Sender security:*** *Π achieves sender security if $\forall \lambda \in \mathbb{N}$, $S_{\mathsf{R}} \subset \{0,1\}^\lambda$, and all pairs $\mathsf{id}_0, \mathsf{id}_1 \in \{0,1\}^\lambda \setminus S_{\mathsf{R}}$ we have that $\mathsf{view}_R^\Pi(S_{\mathsf{R}}, \mathsf{id}_0, \lambda) \approx \mathsf{view}_R^\Pi(S_{\mathsf{R}}, \mathsf{id}_1, \lambda)$. If the distributions are computationally (resp., statistically) indistinguishable then we have computational (resp., statistical) security.*

The PSI functionality can be achieved by casting it as an instance of BP-2PC:

Procedure $\mathsf{SetBP}(S, b_{\mathsf{pth}})$**:**
$\{x_1, \ldots, x_m\} \leftarrow S$; $\lambda \leftarrow |x_1|$; $V, E, T \leftarrow \emptyset$
$V \leftarrow V \cup \{v_\epsilon\}$; $\mathsf{Val}[v_\epsilon] \leftarrow 1$ ▷ set root node
For $1 \le i \le \lambda$ do ▷ add a node for every prefix of length i in S
  For $1 \le j \le m$ do
    $a \leftarrow x_j[1..i]$; $V \leftarrow V \cup \{v_a\}$
    If $|a| = \lambda$ then $\mathsf{Val}[v_a] \leftarrow b_{\mathsf{pth}}$; $T \leftarrow T \cup \{v_a\}$ ▷ accept leaves
    Else $\mathsf{Val}[v_a] \leftarrow |a| + 1$
For all $v_a \in V$ s.t. $|v_a| < \lambda$ do ▷ adding edges from v_a to children
  For $b \in \{0,1\}$ do
    If $\exists\ v_{a\|b} \in V$ then $E \leftarrow E \cup \{(v_a, v_{a\|b}, b)\}$
    Else
      $V \leftarrow V \cup \{v_{a\|b}\}$; $\mathsf{Val}[v_{a\|b}] \leftarrow 1 - b_{\mathsf{pth}}$ ▷ reject leaves
      $E \leftarrow E \cup \{(v_a, v_{a\|b}, b)\}$; $T \leftarrow T \cup \{v_{a\|b}\}$
▷ Pruning: if a node has 2 leaf children with value b_{pth}, delete the children and change parent value to b_{pth}.
While $\exists\ v_a \in V$ s.t. $v_{a\|b} \in T \wedge \mathsf{Val}[v_{a\|b}] = b_{\mathsf{pth}}$ for $b \in \{0,1\}$ do
  $\mathsf{Val}[v_a] \leftarrow b_{\mathsf{pth}}$; $T \leftarrow T \cup \{v_a\}$
  $V \leftarrow V \setminus \{v_{a\|0}, v_{a\|1}\}$; $E \leftarrow E \setminus \{(v_a, v_{a\|b}, b) \mid b \in \{0,1\}\}$
Return (V, E, T, Val)

Fig. 4: Procedure for constructing a BP from a set of m λ-bit strings. See Construction 2. Based on a description in [CGH+21].

1. R runs $\mathsf{SetBP}(S_{\mathsf{R}}, 1)$ (Fig. 4) to generate a branching program $\mathrm{BP}_{\mathrm{psi}}$ such that $\mathsf{BP}_{\mathrm{psi}}(x) = 1$ if $x \in S_{\mathsf{R}}$ and $\mathsf{BP}_{\mathrm{psi}}(x) = 0$ otherwise.
2. R and S run BP-2PC with inputs $\mathrm{BP}_{\mathrm{psi}}$ and id, respectively. By construction of BP-2PC:

$$\begin{cases} R \text{ learns } \mathsf{id} & \text{if } \mathrm{BP}_{\mathrm{psi}}(\mathsf{id}) = 1 \implies \mathsf{id} \in S_{\mathsf{R}} \\ R \text{ does not learn } \mathsf{id} & \text{if } \mathrm{BP}_{\mathrm{psi}}(\mathsf{id}) = 0 \implies \mathsf{id} \notin S_{\mathsf{R}} \end{cases},$$

   which satisfies the PSI correctness condition and security follows from the security of Construction 1 for BP-2PC.

The computation and communication costs of the receiver and sender do not depend on $|S_{\mathsf{R}}|$. If the receiver holds a polynomial-sized BP describing a set S_{R} of exponential size, then this PSI protocol can run in polynomial time.[10]

5.2 Private Set Union (PSU)

As before, assume the sender has a singleton set $S_{\mathsf{S}} = \{\mathsf{id}\}$ where $\mathsf{id} \in \{0,1\}^\lambda$ and the receiver has a set S_{R}. In this setting, we define PSU as follows.

Definition 9 (Private set union (PSU) functionality with $|S_{\mathsf{S}}| = 1$) *Let Π be a two-party communication protocol. Let R be the receiver holding set $S_{\mathsf{R}} \subset \{0,1\}^\lambda$ and let S be the sender holding singleton set $S_{\mathsf{S}} = \{\mathsf{id}\}$, with $\mathsf{id} \in \{0,1\}^\lambda$. Π is a PSU protocol if the following hold after execution of the protocol.*

[10] This assumes R already has the polynomial-sized BP and does not have to build it from their exponential-sized set S_{R}.

- ***Correctness:*** *R learns $S_{\mathsf{R}} \cup \{\mathsf{id}\}$.*
- ***Receiver security:*** *Π achieves receiver security if $\forall \mathsf{id} \in \{0,1\}^\lambda$, and all pairs $S_{\mathsf{R}0}, S_{\mathsf{R}1} \subset \{0,1\}^\lambda$ we have that $\mathsf{view}_S^\Pi(S_{\mathsf{R}0}, \mathsf{id}, \lambda) \approx \mathsf{view}_S^\Pi(S_{\mathsf{R}1}, \mathsf{id}, \lambda)$. If the distributions are computationally (resp., statistically) indistinguishable then we have computational (resp., statistical) security.*
- ***Sender security:*** *Π achieves sender security if $\forall S_{\mathsf{R}} \subset \{0,1\}^\lambda$, and all pairs $\mathsf{id}_0, \mathsf{id}_1 \in S_{\mathsf{R}}$ we have that $\mathsf{view}_R^\Pi(S_{\mathsf{R}}, \mathsf{id}_0, \lambda) \approx \mathsf{view}_R^\Pi(S_{\mathsf{R}}, \mathsf{id}_1, \lambda)$. If the distributions are computationally (resp., statistically) indistinguishable then we have computational (resp., statistical) security.*

The PSU functionality can be achieved by casting it as an instance of BP-2PC:

1. R runs $\mathsf{SetBP}(S_{\mathsf{R}}, 0)$ (Fig. 4) to generate a branching program $\mathrm{BP}_{\mathrm{psu}}$ such that $\mathsf{BP}_{\mathrm{psu}}(x) = 1$ if $x \notin S_{\mathsf{R}}$ and $\mathsf{BP}_{\mathrm{psu}}(x) = 0$ otherwise.
2. R and S run BP-2PC with inputs $\mathrm{BP}_{\mathrm{psu}}$ and id, respectively. By construction of BP-2PC:

$$\begin{cases} R \text{ learns } \mathsf{id} & \text{if } \mathrm{BP}_{\mathrm{psu}}(\mathsf{id}) = 1 \implies \mathsf{id} \notin S_{\mathsf{R}} \\ R \text{ does not learn } \mathsf{id} & \text{if } \mathrm{BP}_{\mathrm{psu}}(\mathsf{id}) = 0 \implies \mathsf{id} \in S_{\mathsf{R}} \end{cases},$$

   which satisfies the PSU correctness condition and security follows from the security of Construction 1 for BP-2PC.

The computation and communication costs of the receiver and sender do not depend on $|S_{\mathsf{R}}|$. If the receiver holds a polynomial-sized BP describing a set S_{R} of exponential size, then this PSU protocol can run in polynomial time.[11]

5.3 Wildcards

Definition 10 (Wildcard) *In a bit string a wildcard, denoted by an asterisk $*$, is used in place of a bit to indicate that its position may hold either bit value. In particular, the wildcard character replaces only a single bit, not a string. (E.g. $00* = \{000, 001\}$ and $**0 = \{000, 010, 100, 110\}$.)*

SetBP in Fig. 4 creates a branching program based on a set that does not contain strings with wildcards. Fig. 5 presents a modified version called SetBP^* which creates a BP based on a singleton set containing a string with wildcard elements. Using SetBP^* instead of SetBP in the constructions for PSI and PSU above allows the receiver's set to contain wildcards.

SetBP^* runs in $O(\lambda)$ time. The resulting BP has depth $\overline{k}$, or $\lambda - k$, where k is the number of wildcard indices, and will contain $2\overline{k} + 1$ nodes, where $\overline{k} \leq \lambda$ is the number of non-wildcard indices. Since the depth leaks the number of wildcards in x, the receiver's message m_1 to the sender in Construction 1 contains the maximum depth $\mathsf{d_m}$, instead of the true depth.

Overview of SetBP^*. SetBP^* (Fig. 5, annotated version in Fig. 10 Appx. A) starts by forming an ordered ascending list of all indices of x without wildcards. Then it loops over each of these indices. A node is added to the BP for every prefix of x ending with an explicit (as opposed to *) bit value. Each node value is set to the index of the next non-wildcard bit in x. The node representing the final non-wildcard index is given value b_{pth}. For example, if $x = 0 * 1 * 0$, then we add prefix nodes $v_\epsilon, v_0, v_{0*1}, v_{0*1*0}$, (where v_ϵ is the root), and set their values to $1, 3, 5, b_{\mathsf{pth}}$, respectively.

Each iteration adds an edge from the previous prefix node to the one just created. This edge is labelled with the bit value at the current non-wildcard index. Continuing with the example, in the iteration that node v_{0*1} is created, an edge from v_0 to v_{0*1} is added with label 1. Since S_{R} only contains one element, we also create a $1 - b_{\mathsf{pth}}$ leaf representing the prefix of the current interior node with the final bit flipped. An edge labelled with this flipped bit is also added from the previous node. In the example, v_{0*0} is created with value $1 - b_{\mathsf{pth}}$ and edge $(v_{0*}, v_{0*0}, 0)$ is added. Once all non-wildcard indices of x have been considered, the BP is returned. If $|S_{\mathsf{R}}| > 1$, SetBP^* can be run multiple times to add more leaf nodes to the BP. After the first SetBP^* run, the zero-set initialization of V, E, T should be omitted.

[11] This assumes R already has the polynomial-sized BP and does not have to build it from their exponential-sized set S_{R}.

Procedure $\mathsf{SetBP}^*(S, b_{\mathsf{pth}})$**:**
$x \leftarrow S$; $\lambda \leftarrow |x|$; $V, E, T \leftarrow \emptyset$; $\overline{\mathsf{WC}} \leftarrow \{i \mid x[i] \neq *\}$; $\overline{k} \leftarrow |\overline{\mathsf{WC}}|$
If $x[1] \neq *$ then $V \leftarrow V \cup \{v_\epsilon\}$; $\mathsf{Val}[v_\epsilon] \leftarrow 1$
If $x[1] = *$ then $V \leftarrow V \cup \{v_\epsilon\}$; $\mathsf{Val}[v_\epsilon] \leftarrow \overline{\mathsf{WC}}[1]$
For $1 \leq i \leq \overline{k}$ do
  $j \leftarrow \overline{\mathsf{WC}}[i]$; $a \leftarrow x[1..j]$; $V \leftarrow V \cup \{v_a\}$
  If $i = \overline{k}$ then $\mathsf{Val}[v_a] \leftarrow b_{\mathsf{pth}}$; $T \leftarrow T \cup \{v_a\}$
  Else $\mathsf{Val}[v_a] \leftarrow \overline{\mathsf{WC}}[i+1]$
  $a_{prev} \leftarrow x[1..\overline{\mathsf{WC}}[i-1]]$; $E \leftarrow E \cup \{(v_{a_{prev}}, v_a, x[j])\}$
  $a' \leftarrow x[1..(j-1)] \,\|\, (1 - x[j])$; $V \leftarrow V \cup \{v_{a'}\}$; $T \leftarrow T \cup \{v_{a'}\}$
  $\mathsf{Val}[v_{a'}] \leftarrow 1 - b_{\mathsf{pth}}$; $E \leftarrow E \cup \{(v_{a_{prev}}, v_{a'}, 1 - x[j])\}$
Return (V, E, T, Val)

Fig. 5: Procedure for constructing a BP from a singleton set with a λ-bit string with wildcards. See Construc. 2 and §5.3. Annotated version in Fig. 10 Appx. A.

5.4 Fuzzy Matching

A fuzzy match [GRS22] in our PSI setting refers to the inclusion of an element $x \in S_{\mathsf{R}}$ in the intersection $S_{\mathsf{R}} \cap S_{\mathsf{S}}$ if S_{S} contains an element that is δ-close to x. The receiver sets a distance threshold δ, which defines an ℓ_∞ ball of radius δ around all points in S_{R}. If an element in S_{S} falls within any of these balls, it counts as a match and the point in S_{R} at the center of this ball will be included in the intersection set. Construction 1 can be used for PSI with fuzzy matches defined with the ℓ_∞-norm as the distance metric (as considered in the structure-aware PSI [GRS22]). This may be accomplished if the receiver's BP can be modified with the addition of wildcards to allow any BP input within an ℓ_∞ ball centred at a point of S_{R} to be accepted as a fuzzy match.

6 Proof of Lemma 3

Proof. (*Condition (1):* $\mathsf{BP}(\mathsf{id}) = 1 \Rightarrow \mathsf{R}_2$ outputs id w.o.p.)
Claim 1: When $\mathsf{DecPath}$ is evaluated on the correct path, it will output (id, r).
Proof of claim 1: Consider the root-to-leaf path of length ℓ induced by the evaluation of $\mathsf{BP}(\mathsf{id})$. By hypothesis $\mathsf{BP}(\mathsf{id}) = 1$, so the path leaf node encodes the value 1. For concreteness, suppose the induced path has the leftmost leaf of the BP, $u_1 \in T$, as the leaf endpoint. With this in mind, denote the path as,

$$\mathsf{pth}[u_1] := (\underbrace{(1^\lambda, x_1, x_1')}_{z_\ell}, \underbrace{(h_1^{(\ell)}, h_2^{(\ell)}, \mathsf{Val}[v_1^{(\ell-1)}])}_{z_{\ell-1}}, \ldots, \underbrace{(h_1^{(1)}, h_2^{(1)}, \mathsf{Val}[v_1^{(0)}])}_{z_0}, \mathsf{h}_{\mathsf{root}}) \,. \tag{1}$$

For the remainder of the proof, node labels v will be identified with their encoded values $\mathsf{Val}[v]$ to save space. Let $(\mathsf{id}_{u_1}, r_{u_1}) \leftarrow \mathsf{DecPath}(\mathsf{pth}[u_1], \widetilde{\mathsf{C}}_{\mathsf{d_m}}, \ldots, \widetilde{\mathsf{C}}_0, \{\mathsf{cth}_{i,b}^{(0)}\})$, where $\widetilde{\mathsf{C}}_{\mathsf{d_m}}, \ldots, \widetilde{\mathsf{C}}_0, \{\mathsf{cth}_{i,b}^{(0)}\}$ are defined as in the construction. Then it suffices to show that $r_{u_1} = r$. Consider an arbitrary iteration $w \in \{0, \ldots, \ell - 2\}$ of the loop in step 2 of $\mathsf{DecPath}$:

2. (a) $\{\mathsf{lb}_i^{(w)}\} \leftarrow \mathsf{HDec}(z_w, \{\mathsf{cth}_{i,b}^{(w)}\})$

$$\leftarrow \mathsf{HDec}((h_1^{(w+1)}, h_2^{(w+1)}, v_1^{(w)}), \mathsf{HEnc}(\mathsf{pp}, h_1^{(w)}, \{\mathsf{lb}_{i,b}^{(w)}\}; \boldsymbol{r}_w))$$

$$\leftarrow \mathsf{HDec}(\underbrace{(h_1^{(w+1)}, h_2^{(w+1)}, v_1^{(w)})}, \mathsf{HEnc}(\mathsf{pp}, \mathsf{Hash}(\mathsf{pp}, \underbrace{(h_1^{(w+1)}, h_2^{(w+1)}, v_1^{(w)})}), \{\mathsf{lb}_{i,b}^{(w)}\}; \boldsymbol{r}_w))$$

Since the two terms indicated are equal, the labels $\{\mathsf{lb}_i^{(w)}\}$ output by HDec are the subset of $\{\mathsf{lb}_{i,b}^{(w)}\}$ corresponding to the bits of $z_w := (h_1^{(w+1)}, h_2^{(w+1)}, v_1^{(w)})$. More precisely, $\mathsf{lb}_{i,0}^{(w)} := \mathsf{lb}_{i,z_w[i]}^{(w)}$ and $\mathsf{lb}_{i,1}^{(w)} := \mathsf{lb}_{i,z_w[i]}^{(w)}$ for all $i \in [n]$.

2. (b) $\{\mathsf{cth}_{i,b}^{(w+1)}\} \leftarrow \mathsf{Eval}(\widetilde{\mathsf{C}}_w, \{\mathsf{lb}_i^{(w)}\})$.

$$\{\mathsf{cth}_{i,b}^{(w+1)}\} \leftarrow \mathsf{V}[\mathsf{pp}, \mathsf{id}, \{\mathsf{lb}_{i,b}^{(w+1)}\}, \boldsymbol{r}_w, r, \mathsf{id}', r', \mathsf{pad}](h_1^{(w+1)}, h_2^{(w+1)}, v_1^{(w)})$$

$$\{\mathsf{cth}_{i,b}^{(w+1)}\} \leftarrow \mathsf{HEnc}(\mathsf{pp}, \underbrace{h_1^{(w+1)}}_{=\mathsf{Hash}(\mathsf{pp},(h_1^{(w+2)},h_2^{(w+2)},v_1^{(w+1)}))}, \{\mathsf{lb}_{i,b}^{(w+1)}\}; \boldsymbol{r}_w) \tag{2}$$

The first input $h_1^{(w+1)}$ is used in the input to HEnc because $\mathsf{pth}[u_1]$ was defined to have the leftmost leaf as an endpoint. In other words, travelling from the root, $\mathsf{pth}[u_1]$ always progresses to the left child.

In the final iteration of the loop, when $w = \ell - 1$, the steps expanded above remain the same except for Equation 2. When $w = \ell - 1$, Eq. 2 is instead

$$\{\mathsf{cth}_{i,b}^{(\ell)}\} \leftarrow \mathsf{HEnc}(\mathsf{pp}, \underbrace{h_1^{(\ell)}}_{=\mathsf{Hash}(\mathsf{pp},(1^\lambda,x_1,x_1'))}, \{\mathsf{lb}_{i,b}^{(\ell)}\}; \boldsymbol{r}_{\ell-1})\,.$$

With this in mind, the final two steps of $\mathsf{DecPath}$ are as follows.

3. $\{\mathsf{lb}_i^{(\ell)}\} \leftarrow \mathsf{HDec}(z_\ell, \{\mathsf{cth}_{i,b}^{(\ell)}\})$

$$\{\mathsf{lb}_i^{(\ell)}\} \leftarrow \mathsf{HDec}(\underbrace{(1^\lambda, x_1, x_1')}, \mathsf{HEnc}(\mathsf{pp}, \mathsf{Hash}(\mathsf{pp}, \underbrace{(1^\lambda, x_1, x_1')}), \{\mathsf{lb}_{i,b}^{(\ell)}\}))$$

Since the two terms indicated above are equal, the labels $\{\mathsf{lb}_i^{(\ell)}\}$ output by HDec are the subset of labels $\{\mathsf{lb}_{i,b}^{(\ell)}\}$ used in the input to HEnc, where the subset corresponds to the bits of $z_\ell = (1^\lambda, x_1, x_1')$.

4. $\{\mathsf{id}_{u_1}, r_{u_1}, \mathsf{pad}\} \leftarrow \mathsf{Eval}(\widetilde{\mathsf{C}}_\ell, \{\mathsf{lb}_i^{(\ell)}\})$

$$\{\mathsf{id}_{u_1}, r_{u_1}, \mathsf{pad}\} \leftarrow \mathsf{V}[\mathsf{pp}, \mathsf{id}, \{\mathsf{lb}_{i,b}^{(\ell+1)}\}, \boldsymbol{r}_\ell, r, \mathsf{id}', r', \mathsf{pad}](1^\lambda, x_1, x_1')$$

$$\{\mathsf{id}_{u_1}, r_{u_1}, \mathsf{pad}\} \leftarrow \{\mathsf{id}_{u_1} \leftarrow \mathsf{id}, r_{u_1} \leftarrow r, \text{ and } \mathsf{pad} \xleftarrow{\$} \{0,1\}^{2(n-1)}\}$$

Then return $(\mathsf{id}_{u_1}, r_{u_1})$ to the receiver. The first input to V is 1^λ, so the tuple $(\mathsf{id}_{u_1}, r_{u_1})$ is equal to (id, r).

The receiver compares r_{u_1} from $\mathsf{DecPath}$ with r in m_2. Since these strings are equal, the receiver takes id_{u_1} output from $\mathsf{DecPath}$ as the sender's element. Hence, the receiver learns id when $\mathsf{BP}(\mathsf{id}) = 1$, completing the proof of claim 1.

In the above, we made use of the correctness properties of garbled circuit evaluation and HE decryption. These guarantees give us that $\Pr[\mathsf{id}_{u_1} = \mathsf{id} \wedge r_{u_1} = r \mid (\mathsf{id}_{u_1}, r_{u_1}) \leftarrow \mathsf{DecPath}(\mathsf{pth}[u_1], \mathsf{m}_2)] = 1$ when $\mathsf{pth}[u_1]$ is the correct path through the BP. In order for the correctness condition (1) to be met, it must also be true that there does not exist any other path $\mathsf{pth}[u'] \neq \mathsf{pth}[u_1]$ such that $r_{u'} = r$ where $(\mathsf{id}_{u'}, r_{u'}) \leftarrow \mathsf{DecPath}(\mathsf{pth}[u'], \mathsf{m}_2)$. In other words, there must not exist an incorrect path that decrypts the correct signal value r.

Claim 2: With at most negligible probability, there exists an incorrect path that when input to $\mathsf{DecPath}$, decrypts to the correct signal value r.
Proof of claim 2: To show that occurs with negligible probability, consider running $\mathsf{DecPath}$ on an incorrect path $\mathsf{pth}[u'] \neq \mathsf{pth}[u_1]$. Let $\mathsf{pth}[u_1]$ and $\mathsf{pth}[u']$ have lengths ℓ and ℓ', respectively where $1 \leq \ell, \ell' \leq d$. Suppose these paths are equal at level $\alpha - 1$ and differ at level α onward, for some $\alpha \in \{0, \ldots, \min\{\ell, \ell'\}\}$. Suppose $u_1 \in T$ is the leftmost leaf, as above, and $u' \in T \setminus \{u_1\}$ is the leaf endpoint of $\mathsf{pth}[u']$. Let these paths be given by the following.

$$\mathsf{pth}[u_1] := (\underbrace{(u_1^{(\ell)\times\lambda}, x_1, x_1')}_{z_\ell}, \underbrace{(h_1^{(\ell)}, h_2^{(\ell)}, v_1^{(\ell-1)})}_{z_{\ell-1}}, \ldots, \underbrace{(h_1^{(\alpha+1)}, h_2^{(\alpha+1)}, v_1^{(\alpha)})}_{z_\alpha}, \underbrace{(h_1^{(\alpha)}, h_2^{(\alpha)}, v_1^{(\alpha-1)})}_{z_{\alpha-1}}, \ldots, \underbrace{(h_1^{(1)}, h_2^{(1)}, v_1^{(0)})}_{z_0}, \mathsf{h}_{\mathsf{root}}) \tag{3}$$

$$\mathsf{pth}[u'] := (\underbrace{(u'^{(\ell')\times\lambda}, x, x')}_{z'_{\ell'}}, \underbrace{(h^{(\ell')}, h'^{(\ell')}, v^{(\ell'-1)})}_{z'_{\ell'-1}}, \ldots, \underbrace{(h_3^{(\alpha+1)}, h_4^{(\alpha+1)}, v_2^{(\alpha)})}_{z'_{\alpha}}, \underbrace{(h_1^{(\alpha)}, h_2^{(\alpha)}, v_1^{(\alpha-1)})}_{z'_{\alpha-1}}, \ldots, \underbrace{(h_1^{(1)}, h_2^{(1)}, v_1^{(0)})}_{z'_0}, \mathsf{h}_{\mathsf{root}}) . \tag{4}$$

Since $\mathsf{pth}[u']$ differs from the correct path at level α, the steps of $\mathsf{DecPath}(\mathsf{pth}[u'], \mathsf{m}_2)$ and $\mathsf{DecPath}(\mathsf{pth}[u_1], \mathsf{m}_2)$ will be identical until loop iteration $w = \alpha$. Consider iteration $w = \alpha - 1$ of $\mathsf{DecPath}(\mathsf{pth}[u'], \mathsf{m}_2)$:

2. (a) $\{\mathsf{lb}_i^{(\alpha-1)}\} \leftarrow \mathsf{HDec}(z'_{\alpha-1}, \{\mathsf{cth}_{i,b}^{(\alpha-1)}\})$

$$\leftarrow \mathsf{HDec}((h_1^{(\alpha)}, h_2^{(\alpha)}, v_1^{(\alpha-1)}), \mathsf{HEnc}(\mathsf{pp}, h_1^{(\alpha-1)}, \{\mathsf{lb}_{i,b}^{(\alpha-1)}\}))$$
$$\leftarrow \mathsf{HDec}(\underbrace{(h_1^{(\alpha)}, h_2^{(\alpha)}, v_1^{(\alpha-1)})}, \mathsf{HEnc}(\mathsf{pp}, \mathsf{Hash}(\mathsf{pp}, \underbrace{(h_1^{(\alpha)}, h_2^{(\alpha)}, v_1^{(\alpha-1)})}), \{\mathsf{lb}_{i,b}^{(\alpha-1)}\})) .$$

Since the indicated terms are equal, the $\{\mathsf{lb}_i^{(\alpha-1)}\}$ labels output are the labels of circuit $\widetilde{\mathsf{C}}_{\alpha-1}$ corresponding to the bits of $z'_{\alpha-1}$.

2. (b) $\{\mathsf{cth}_{i,b}^{(\alpha)}\} \leftarrow \mathsf{Eval}(\widetilde{\mathsf{C}}_{\alpha-1}, \{\mathsf{lb}_i^{(\alpha-1)}\})$

$$\leftarrow \mathsf{V}[\mathsf{pp}, \mathsf{id}, \{\mathsf{lb}_{i,b}^{(\alpha)}\}, \boldsymbol{r}_{\alpha-1}, r, \mathsf{id}', r', \mathsf{pad}](h_1^{(\alpha)}, h_2^{(\alpha)}, v_1^{(\alpha-1)})$$
$$\leftarrow \mathsf{HEnc}(\mathsf{pp}, h_1^{(\alpha)}, \{\mathsf{lb}_{i,b}^{(\alpha)}\}; \boldsymbol{r}_{\alpha-1}) .$$

In the last line, $h_1^{(\alpha)}$ is used in the hash encryption due to the assumption that the correct path has the leftmost leaf as an endpoint, meaning $\mathsf{id}[v_1^{(\alpha+1)}] = 0$.[12] Next, the $w = \alpha$ iteration of the loop proceeds as follows.

2. (a) $\{\mathsf{lb}_i'^{(\alpha)}\} \leftarrow \mathsf{HDec}(z'_{\alpha}, \{\mathsf{cth}_{i,b}^{(\alpha)}\})$

$$\leftarrow \mathsf{HDec}((h_3^{(\alpha+1)}, h_4^{(\alpha+1)}, v_2^{(\alpha)}), \mathsf{HEnc}(\mathsf{pp}, h_1^{(\alpha)}, \{\mathsf{lb}_{i,b}^{(\alpha)}\}; \boldsymbol{r}_{\alpha-1}))$$
$$\leftarrow \mathsf{HDec}(\underbrace{(h_3^{(\alpha+1)}, h_4^{(\alpha+1)}, v_2^{(\alpha)})}, \mathsf{HEnc}(\mathsf{pp}, \underbrace{(h_1^{(\alpha+1)}, h_2^{(\alpha+1)}, v_1^{(\alpha)})}, \{\mathsf{lb}_{i,b}^{(\alpha)}\}; \boldsymbol{r}_{\alpha-1})) .$$

The two indicated terms are not equal. Decrypting an HE ciphertext with an incorrect hash preimage produces an output containing no PPT-accessible information about the encrypted plaintext. For this reason, a prime was added above to the LHS labels to differentiate them from the labels encrypted on the RHS. Thus $\{\mathsf{lb}_i'^{(\alpha)}\}$ provides no information about $\{\mathsf{lb}_{i,b}^{(\alpha)}\}$.

2. (b) $\{\mathsf{cth}_{i,b}^{(\alpha+1)}\} \leftarrow \mathsf{Eval}(\widetilde{\mathsf{C}}_{\alpha}, \{\mathsf{lb}_i'^{(\alpha)}\})$.

Note that $\{\mathsf{lb}_i'^{(\alpha)}\}$ are *not* labels of $\widetilde{\mathsf{C}}_{\alpha}$, and certainly not a subset of those labels corresponding to a meaningful input. So the output $\{\mathsf{cth}_{i,b}^{(\alpha+1)}\}$ is a meaningless set of strings, not a ciphertext.

For w from α to ℓ', every HDec operation will output $\{\mathsf{lb}_i'^{(w)}\}$ which are *not* circuit labels for $\widetilde{\mathsf{C}}_w$ and every evaluation $\mathsf{Eval}(\widetilde{\mathsf{C}}_w, \{\mathsf{lb}_i'^{(w)}\})$ will output strings with no relation to $\widetilde{\mathsf{C}}_w$. In step 4, $\{\mathsf{id}_{u'}, r_{u'}, \mathsf{pad}\} \leftarrow \mathsf{Eval}(\widetilde{\mathsf{C}}_{\ell'}, \{\mathsf{lb}_i'^{(\ell')}\})$ is computed. Since $\{\mathsf{lb}_i'^{(\ell')}\}$ are not labels, the evaluation output is meaningless. In particular, the tuple $(\mathsf{id}_{u'}, r_{u'})$ output to R_2 contains no PPT-accessible information about (id, r). Hence $\Pr[r_{u'} = r] \leq 2^{-\lambda} + \mathsf{negl}(\lambda)$. By assumption on the size of BP, there are a polynomial number of root-to-leaf paths, thus by the union bound the probability that any incorrect root-to-path causes $\mathsf{DecPath}$ to output r is,

$$\Pr[\exists\ u \in T \setminus \{u_1\} \text{ s.t. } r_u = r \mid (\mathsf{id}_u, r_u) \leftarrow \mathsf{DecPath}(\mathsf{pth}[u], \mathsf{m}_2)] \leq \frac{\mathsf{poly}(\lambda)}{2^{\lambda}} + \mathsf{negl}(\lambda) .$$

[12] It is straightforward to change the proof to apply to cases of different correct paths.

The probability that R_2 outputs id when $\mathsf{BP}(\mathsf{id}) = 1$ is the probability that none of the incorrect paths output a signal value equal to r:

$$\Pr[\mathsf{R}_2 \text{ outputs } \mathsf{id} \mid \mathsf{BP}(\mathsf{id}) = 1] \geq 1 - \frac{\mathsf{poly}(\lambda)}{2^\lambda} - \mathsf{negl}(\lambda)\,.$$

Thus proving claim 2 and correctness condition (1).

(*Condition (2):* $\mathsf{BP}(\mathsf{id}) = 0 \Rightarrow \mathsf{R}_2$ outputs $\perp$ w.o.p.) In the proof of Theorem 1 we will show that when $\mathsf{BP}(\mathsf{id}) = 0$,

$$(\widetilde{\mathsf{C}}_{\mathsf{d_m}}, \ldots, \widetilde{\mathsf{C}}_0, \{\mathsf{cth}_{i,b}^{(0)}\}, r) \overset{c}{\equiv} (\widetilde{\mathsf{C}}'_{\mathsf{d_m}}, \ldots, \widetilde{\mathsf{C}}'_0, \{\mathsf{cth}_{i,b}^{\prime(0)}\}, r')\,, \tag{5}$$

where all primed values are sampled uniformly random. On the LHS, the circuits $\widetilde{\mathsf{C}}_{\mathsf{d_m}}, \ldots, \widetilde{\mathsf{C}}_0$ all have the signal value r hardcoded, while the RHS is independent of r. So, for all fixed $u \in T$,

$$\Pr\Big[r_u = r \mid (\mathsf{id}_u, r_u) \leftarrow \mathsf{DecPath}(\mathsf{pth}[u], (\widetilde{\mathsf{C}}'_{\mathsf{d_m}}, \ldots, \widetilde{\mathsf{C}}'_0, \{\mathsf{cth}_{i,b}^{\prime(0)}\}, r'))\Big] \leq \frac{1}{2^\lambda}\,,$$

where $\mathsf{pth}[u]$ denotes the path from the root to leaf u. By assumption on the size of BP, there are a polynomial number of root-to-leaf paths, thus by the union bound the probability that any root-to-leaf paths decrypt to output r is,

$$\Pr\Big[\exists\, u \in T \text{ s.t. } r_u = r \mid (\mathsf{id}_u, r_u) \leftarrow \mathsf{DecPath}(\mathsf{pth}[u], (\widetilde{\mathsf{C}}'_{\mathsf{d_m}}, \ldots, \widetilde{\mathsf{C}}'_0, \{\mathsf{cth}_{i,b}^{\prime(0)}\}, r'))\Big] \leq \frac{\mathsf{poly}(\lambda)}{2^\lambda}\,.$$

By Equation 5, we must also have that the analogous probability for inputs $(\widetilde{\mathsf{C}}_{\mathsf{d_m}}, \ldots, \widetilde{\mathsf{C}}_0, \{\mathsf{cth}_{i,b}^{(0)}\}, r)$ is computationally indistinguishable. Thus,

$$\Pr\Big[\exists\, u \in T \text{ s.t. } r_u = r \mid (\mathsf{id}_u, r_u) \leftarrow \mathsf{DecPath}(\mathsf{pth}[u], (\widetilde{\mathsf{C}}_{\mathsf{d_m}}, \ldots, \widetilde{\mathsf{C}}_0, \{\mathsf{cth}_{i,b}^{(0)}\}, r))\Big] \leq \frac{\mathsf{poly}(\lambda)}{2^\lambda} + \mathsf{negl}(\lambda)\,.$$

If R_2 receives r_u from $\mathsf{DecPath}$ such that $r_u = r$, then R_2 outputs id_u, not $\perp$. It directly follows that,

$$\Pr[\mathsf{R}_2 \text{ does not output } \perp \mid \mathsf{BP}(\mathsf{id}) = 0] \leq \frac{\mathsf{poly}(\lambda)}{2^\lambda} + \mathsf{negl}(\lambda)$$

$$\Pr[\mathsf{R}_2 \text{ outputs } \perp \mid \mathsf{BP}(\mathsf{id}) = 0] \geq 1 - \frac{\mathsf{poly}(\lambda)}{2^\lambda} - \mathsf{negl}(\lambda)\,,$$

which completes the proof of Lemma 3. □

7 Proof of Theorem 1

*Proof (**Theorem 1 receiver security proof.**).* Note that node labels will be identified with their encoded values to save space. Following Definition 7, for any pair $(\mathrm{BP}_0, \mathrm{BP}_1)$ consider the distribution below for $i \in \{0, 1\}$.

$$\begin{aligned}\mathsf{view}_S^{\mathsf{BP\text{-}2PC}}(\mathrm{BP}_i, \mathsf{id}, \lambda) &= (\mathsf{id}, \boldsymbol{r}_\mathsf{S}, \mathsf{m}_1, \mathsf{m}_2) \\ &= (\mathsf{id}, \boldsymbol{r}_\mathsf{S}, (\mathsf{d_m}, \mathsf{h}_{\mathsf{root}\,i}), (\widetilde{\mathsf{C}}_{\mathsf{d_m}}, \ldots, \widetilde{\mathsf{C}}_0, \mathsf{HEnc}(\mathsf{pp}, \mathsf{h}_{\mathsf{root}\,i}, \{\mathsf{lb}_{i,b}^{(0)}\}), r))\,,\end{aligned}$$

where $\boldsymbol{r}_\mathsf{S}$ are the sender's random coins, $\mathsf{h}_{\mathsf{root}\,i}$ is the root hash, and $\mathsf{d_m}$ is the maximum depth of branching program BP_i. Since both BPs have security parameter λ, both will have $\mathsf{d_m} = \lambda + 1$. Let d_i be the depth of BP_i.

Robustness of HE implies that for all $\mathsf{pp} \overset{\$}{\leftarrow} \mathsf{HGen}(1^\lambda, 3\lambda)$ and $u \in T$, the distribution $(\mathsf{pp}, \mathsf{Hash}(\mathsf{pp}, (u^\lambda, x, x')))$, where $x, x' \overset{\$}{\leftarrow} \{0,1\}^\lambda$, is statistically close to $(\mathsf{pp}, h_\$)$ where $h_\$ \overset{\$}{\leftarrow} \{0,1\}^\lambda$. Hence $\mathsf{Hash}(\mathsf{pp}, (u^\lambda, x, x'))$ statistically hides u. At level d_i, BP_i will have at least two leaf nodes with the same parent. Let u_1, u_2 be two such leaves and let $v^{(d_i-1)}$ be the parent. Node $v^{(d_i-1)}$ will then have hash value,

$$h^{(d_i-1)} \leftarrow \mathsf{Hash}(\mathsf{pp}, (h_1^{(d_i)}, h_2^{(d_i)}, v^{(d_i-1)}))$$

$$\leftarrow \mathsf{Hash}(\mathsf{pp}, (\mathsf{Hash}(\mathsf{pp}, (u_1^\lambda, x_1, x_1')), \mathsf{Hash}(\mathsf{pp}, (u_2^\lambda, x_2, x_2')), v^{(d_i-1)}))\,.$$

Since $h_1^{(d_i)}$ and $h_2^{(d_i)}$ are both statistically close to uniform, we have that $h^{(d_i-1)}$ is also statistically close to uniform. Continuing up the tree in this way, we see that the root hash $\mathsf{h}_{\mathsf{root}_i}$ is also indistinguishable from random. Thus $\mathsf{h}_{\mathsf{root}_0} \approx_s \mathsf{h}_{\mathsf{root}_1}$, which gives us $\mathsf{view}_S^{\mathsf{BP\text{-}2PC}}(\mathrm{BP}_0, \mathsf{id}, \lambda) \approx_s \mathsf{view}_S^{\mathsf{BP\text{-}2PC}}(\mathrm{BP}_1, \mathsf{id}, \lambda)$. □

*Proof (**Theorem 1 sender security proof**).*
Sender security will be proved through a sequence of indistinguishable hybrids in two main steps. First, all garbled circuits in the sender's message m_2 are replaced one at a time with simulated circuits. Then m_2 is switched to random.

Sender security only applies when $\mathsf{BP}(\mathsf{id}) = 0$, so this will be assumed for the proof. For concreteness, suppose the path induced on the BP by evaluating id has the leftmost leaf as an endpoint. In particular, let

$$\mathsf{pth} := (\underbrace{(\mathsf{Val}[v_1^{(\ell)}]^{\times\lambda}, x_1, x_1')}_{z_\ell}, \underbrace{(h_1^{(\ell)}, h_2^{(\ell)}, \mathsf{Val}[v_1^{(\ell-1)}])}_{z_{\ell-1}}, \ldots, \underbrace{(h_1^{(1)}, h_2^{(1)}, \mathsf{Val}[v_1^{(0)}])}_{z_0}, \mathsf{h}_{\mathsf{root}}) \tag{6}$$

be the leaf-root path induced on the hashed BP by evaluation of id, where ℓ is the path length and d is the BP depth.[13] Since $\mathsf{BP}(\mathsf{id}) = 0$, the terminal node encodes value 0, i.e, $\mathsf{Val}[v_1^{(\ell)}] = 0$. We let $\mathsf{h}_{\mathsf{root}} \leftarrow \mathsf{Hash}(\mathsf{pp}, z_0)$ and $h_1^{(i)} \leftarrow \mathsf{Hash}(\mathsf{pp}, z_i)$ for all $1 \le i \le \ell$, where the z_i values are defined as in Eq. 6. To save space, often node labels v will be identified with their encoded index values $\mathsf{Val}[v]$ and the padding superscript will be omitted from leaf node values.

Hyb$_0$: [Fig. 6 left] The sender's message $\mathsf{m}_2 := (\widetilde{\mathsf{C}}_{\mathsf{d_m}}, \ldots, \widetilde{\mathsf{C}}_0, \{\mathsf{cth}_{i,b}^{(0)}\}, r)$ is formed as described in the construction.

Hyb$_1$: [Fig. 6 right] All circuits are simulated. The circuits are simulated so that if R runs $\mathsf{DecPath}$ on pth with simulated circuits, then every step occurs, from the view of R, as it would in **Hyb**$_0$. This requires knowledge of pth and the correct sequence of hash preimages $z_\ell, \ldots, z_0$, where $z_\ell = (0^\lambda, x_1, x_1')$ and $z_j = (h_1^{(j+1)}, h_2^{(j+1)}, v_1^{(j)})$ for $j \in \{0, \ldots, \ell-1\}$. By assumption of pth, every evaluation $\mathsf{Eval}(\widetilde{\mathsf{C}}_j, \{\mathsf{lb}_i^{(j)}\})$, where $\{\mathsf{lb}_i^{(j)}\} \leftarrow \mathsf{HDec}(z_j, \{\mathsf{cth}_{i,b}^{(j)}\})$, done in $\mathsf{DecPath}$ for $j \in \{0, \ldots, \ell-1\}$ will output ciphertexts $\mathsf{HEnc}(\mathsf{pp}, h_1^{(j+1)}, \{\mathsf{lb}_{i,b}^{(j+1)}\}; \boldsymbol{r}_j)$[14]. Moreover, evaluation of $\mathsf{Eval}(\widetilde{\mathsf{C}}_\ell, \{\mathsf{lb}_i^{(\ell)}\})$ outputs $\{\mathsf{id}', r', \mathsf{pad}\}$ for random $\mathsf{id}', r' \xleftarrow{\$} \{0,1\}^\lambda$ and $\mathsf{pad} \xleftarrow{\$} \{0,1\}^{2(n-1)}$. Simulating circuits $\widetilde{\mathsf{C}}_\ell, \ldots, \widetilde{\mathsf{C}}_0$ is straightforward.

To simulate circuits $\widetilde{\mathsf{C}}_{\mathsf{d_m}}, \ldots, \widetilde{\mathsf{C}}_{\ell+1}$ note that none of these circuits can be used by R in $\mathsf{DecPath}$ to obtain a meaningful output. Only this behaviour needs to be mimicked. To this end, we define "ghost" values $z_{\mathsf{d_m}}, \ldots, z_{\ell+1}$ with their associated hash values. The deepest is defined to be uniformly random: $z_{\mathsf{d_m}} \xleftarrow{\$} \{0,1\}^{3\lambda}$. Then for $j \in \{\mathsf{d_m} - 1, \ldots, \ell+1\}$ define,

$$h'^{(j)} := \mathsf{Hash}(\mathsf{pp}, (\overbrace{\underbrace{h'^{(j+1)}}_{\mathsf{Hash}(\mathsf{pp}, z_{j+1})}, h'^{(j+1)}, v'^{(j)}}^{z_j}))$$

where $v_j' \xleftarrow{\$} \{0,1\}^\lambda$. In this way, two-thirds of the z_j preimage is uniformly random which allows us to invoke the HE robustness property. Moreover, the z_j values create a chain of preimages similar to the z_j values for $0 \le j \le \ell - 1$.

Lemma 4 *Hybrids* **Hyb**$_0$ *and* **Hyb**$_1$ *are computationally indistinguishable.*

Hyb$_2$: Sample m_2 at random.

Lemma 5 *Hybrids* **Hyb**$_1$ *and* **Hyb**$_2$ *are computationally indistinguishable.*

If m_2 is pseudorandom to the receiver, then m_2 created with some id_0 is computationally indistinguishable from m_2 created with some other id_1. Therefore we have $\mathsf{view}_R^{\mathsf{BP\text{-}2PC}}(\mathrm{BP}, \mathsf{id}_0, \lambda) \stackrel{c}{\equiv} \mathsf{view}_R^{\mathsf{BP\text{-}2PC}}(\mathrm{BP}, \mathsf{id}_1, \lambda)$, hence the above two lemmas establish computational sender security.

[13] We assume $\ell \ge 1$. If the receiver's BP has depth 0, then two dummy leaves can be introduced as root children.

[14] The use of $h_1^{(j+1)}$ in HEnc is from the assumption that pth has the leftmost leaf as an endpoint and hence the first hash input is always used in the V encryption. In the general case, this hash value would be changed accordingly.

$\mathbf{Hyb}_0$:
$r, r', \mathsf{id}' \xleftarrow{\$} \{0,1\}^\lambda$; $\mathsf{pad} \xleftarrow{\$} \{0,1\}^{2(n-1)}$
$\mathsf{C}_{\mathsf{d_m}} := \mathsf{F}[\mathsf{id}, \mathsf{id}', r, r']$
$(\widetilde{\mathsf{C}}_{\mathsf{d_m}}, \{\mathsf{lb}_{i,b}^{(\mathsf{d_m})}\}) \xleftarrow{\$} \mathsf{Garb}(\mathsf{C}_{\mathsf{d_m}})$
For w from $\mathsf{d_m} - 1$ to 0 do
  Sample random $\boldsymbol{r}_w$
  $\mathsf{C}_w := \mathsf{V}[\mathsf{pp}, \mathsf{id}, \{\mathsf{lb}_{i,b}^{(w+1)}\}, \boldsymbol{r}_w, r, \mathsf{id}', r', \mathsf{pad}]$
  $(\widetilde{\mathsf{C}}_w, \{\mathsf{lb}_{i,b}^{(w)}\}) \xleftarrow{\$} \mathsf{Garb}(\mathsf{C}_w)$
$\{\mathsf{cth}_{i,b}^{(0)}\} \xleftarrow{\$} \mathsf{HEnc}(\mathsf{pp}, \mathsf{h_{root}}, \{\mathsf{lb}_{i,b}^{(0)}\})$
$\mathsf{m}_2 := (\widetilde{\mathsf{C}}_{\mathsf{d_m}}, \ldots, \widetilde{\mathsf{C}}_0, \{\mathsf{cth}_{i,b}^{(0)}\}, r)$
Return m_2

$\mathbf{Hyb}_1$:
$r, r', \mathsf{id}' \xleftarrow{\$} \{0,1\}^\lambda$; $\mathsf{pad} \xleftarrow{\$} \{0,1\}^{2(n-1)}$
$z_{\mathsf{d_m}} \xleftarrow{\$} \{0,1\}^{3\lambda}$; $(\widetilde{\mathsf{C}}_{\mathsf{d_m}}, \{\mathsf{lb}_i^{(\mathsf{d_m})}\}) \xleftarrow{\$} \mathsf{Sim}(\mathsf{F}, \{\mathsf{id}', r'\})$
For $0 \le w \le \mathsf{d_m} - 1$ sample random $\boldsymbol{r}_w$
For i from $\mathsf{d_m} - 1$ down to $\ell + 1$ do
  $v'^{(i)} \xleftarrow{\$} \{0,1\}^\lambda$; $h'^{(i+1)} \leftarrow \mathsf{Hash}(\mathsf{pp}, z_{i+1})$
  $z_i := (h'^{(i+1)}, h'^{(i+1)}, v'^{(i)})$
For w from $\mathsf{d_m} - 1$ down to $\ell + 1$ do
  $\{\mathsf{cth}_{i,b}^{(w+1)}\} \leftarrow \mathsf{HEnc}(\mathsf{pp}, h'^{(w+1)}, \{\mathsf{lb}_{i,b}^{(w+1)}\}; \boldsymbol{r}_w)$
  $(\widetilde{\mathsf{C}}_w, \{\mathsf{lb}_i^{(w)}\}) \xleftarrow{\$} \mathsf{Sim}(\mathsf{V}, \{\mathsf{cth}_{i,b}^{(w+1)}\})$
$(\widetilde{\mathsf{C}}_\ell, \{\mathsf{lb}_i^{(\ell)}\}) \xleftarrow{\$} \mathsf{Sim}(\mathsf{V}, \{\mathsf{id}', r', \mathsf{pad}\})$
For w from $\ell - 1$ down to 0 do
  $\{\mathsf{cth}_{i,b}^{(w+1)}\} \leftarrow \mathsf{HEnc}(\mathsf{pp}, h_1^{(w+1)}, \{\mathsf{lb}_i^{(w+1)}\}; \boldsymbol{r}_w)$
  $(\widetilde{\mathsf{C}}_w, \{\mathsf{lb}_i^{(w)}\}) \xleftarrow{\$} \mathsf{Sim}(\mathsf{V}, \{\mathsf{cth}_{i,b}^{(w+1)}\})$
$\{\mathsf{cth}_{i,b}^{(0)}\} \xleftarrow{\$} \mathsf{HEnc}(\mathsf{pp}, \mathsf{h_{root}}, \{\mathsf{lb}_i^{(0)}\})$
Return $\mathsf{m}_2 := (\widetilde{\mathsf{C}}_{\mathsf{d_m}}, \ldots, \widetilde{\mathsf{C}}_0, \{\mathsf{cth}_{i,b}^{(0)}\}, r)$

Fig. 6: $\mathbf{Hyb}_0$ and $\mathbf{Hyb}_1$ for the proof of Theorem 1.

7.1 Proof of Lemma 4

To prove that $\mathbf{Hyb}_0 \stackrel{c}{\equiv} \mathbf{Hyb}_1$, we define a chain of $\mathsf{d_m} + 1$ hybrids between $\mathbf{Hyb}_0$ and $\mathbf{Hyb}_1$. Then we prove each game hop is indistinguishable.

$\mathbf{Hyb}_{1.p}$ *for* $0 \le p \le \mathsf{d_m}$ *(Fig. 8):* Let pth be as in Eq. 6 and recall we assume that $\mathsf{Val}[v_1^{(\ell)}] = 0$. In $\mathbf{Hyb}_{1.p}$ circuits $\widetilde{\mathsf{C}}_0, \ldots, \widetilde{\mathsf{C}}_{p-1}$ are simulated and circuits $\widetilde{\mathsf{C}}_p, \ldots, \widetilde{\mathsf{C}}_{\mathsf{d_m}}$ are honestly generated (as in $\mathbf{Hyb}_0$). In $\mathbf{Hyb}_{1.0}$, all circuits are generated honestly[15] and in $\mathbf{Hyb}_{1.\mathsf{d_m}}$ all circuits are simulated except for $\widetilde{\mathsf{C}}_{\mathsf{d_m}}$. The way a particular circuit $\widetilde{\mathsf{C}}_i$ for $i \le p-1$ is simulated depends on if $i < \ell$, $i = \ell$, or $i > \ell$, where ℓ is the length of path induced by id. These differences are shown in Fig. 7. As in $\mathbf{Hyb}_1$, simulating circuits $\widetilde{\mathsf{C}}_{\ell+1}, \ldots, \widetilde{\mathsf{C}}_{\mathsf{d_m}-1}$ is done using ciphertexts created with "ghost" z values.

Lemma 6 $\mathbf{Hyb}_0 \stackrel{c}{\equiv} \mathbf{Hyb}_{1.0}$ *and* $\mathbf{Hyb}_1 \stackrel{c}{\equiv} \mathbf{Hyb}_{1.\mathsf{d_m}}$.

Proof. First we will prove $\mathbf{Hyb}_0 \stackrel{c}{\equiv} \mathbf{Hyb}_{1.0}$ (Fig. 6 and Fig. 8). In both hybrids all circuits are honestly generated, but they differ in two ways. The first is in how the labels $\{\mathsf{lb}^{(\mathsf{d_m})}\}$ are formed. Both hybrids generate the tuple $(\widetilde{\mathsf{C}}_{\mathsf{d_m}}, \{\mathsf{lb}_{i,b}^{(\mathsf{d_m})}\}) \xleftarrow{\$} \mathsf{Garb}(\mathsf{F}[\mathsf{id}, \mathsf{id}', r, r'])$ but $\mathbf{Hyb}_{1.0}$ additionally does $\{\mathsf{lb}_i^{(\mathsf{d_m})}\} := \{\mathsf{lb}_{i,z_{\mathsf{d_m}}[i]}^{(\mathsf{d_m})}\}$. If $\ell < \mathsf{d_m}$, then $z_{\mathsf{d_m}}$ is random. In that case, $\mathsf{Eval}(\widetilde{\mathsf{C}}_{\mathsf{d_m}}, \{\mathsf{lb}_{i,z_{\mathsf{d_m}}[i]}^{(\mathsf{d_m})}\})$ will return $\{\mathsf{id}', r'\}$ w.o.p. If $\ell = \mathsf{d_m}$ then $z_{\mathsf{d_m}} := (0^\lambda, x_1, x_1')$ and so $\mathsf{Eval}(\widetilde{\mathsf{C}}_{\mathsf{d_m}}, \{\mathsf{lb}_{i,z_{\mathsf{d_m}}[i]}^{(\mathsf{d_m})}\})$ will return $\{\mathsf{id}', r'\}$ with probability 1. Hence the difference between the sets of labels is indistinguishable by the $\mathsf{BP}(\mathsf{id}) = 0$ assumption.

The second difference between $\mathbf{Hyb}_0$ and $\mathbf{Hyb}_{1.0}$ is in how $\{\mathsf{cth}_{i,b}^{(0)}\}$ is formed. In $\mathbf{Hyb}_0$ we define $\{\mathsf{cth}_{i,b}^{(0)}\} \xleftarrow{\$} \mathsf{HEnc}(\mathsf{pp}, \mathsf{h_{root}}, \{\mathsf{lb}_{i,b}^{(0)}\})$. While $\mathbf{Hyb}_{1.0}$ does $\{\mathsf{cth}_{i,b}^{(0)}\} \xleftarrow{\$} \mathsf{HEnc}(\mathsf{pp}, \mathsf{h_{root}}, \{\mathsf{lb}_i^{(0)}\})$, where $\{\mathsf{lb}_i^{(0)}\} := \{\mathsf{lb}_{i,z_0[i]}^{(0)}\}$. Since $\mathsf{h_{root}} \leftarrow \mathsf{Hash}(\mathsf{pp}, z_0)$, by semantic security of hash encryption we have that $\mathsf{HEnc}(\mathsf{pp}, \mathsf{h_{root}}, \{\mathsf{lb}_i^{(0)}\}) \stackrel{c}{\equiv} \mathsf{HEnc}(\mathsf{pp}, \mathsf{h_{root}}, \{\mathsf{lb}_{i,b}^{(0)}\})$, completing the proof of $\mathbf{Hyb}_0 \stackrel{c}{\equiv} \mathbf{Hyb}_{1.0}$.

[15] When $p = 0$, $\mathbf{Hyb}_{1.p}$ is defined so that circuits $\widetilde{\mathsf{C}}_0, \ldots, \widetilde{\mathsf{C}}_{-1}$ are simulated, which we define to mean that no circuits are simulated.

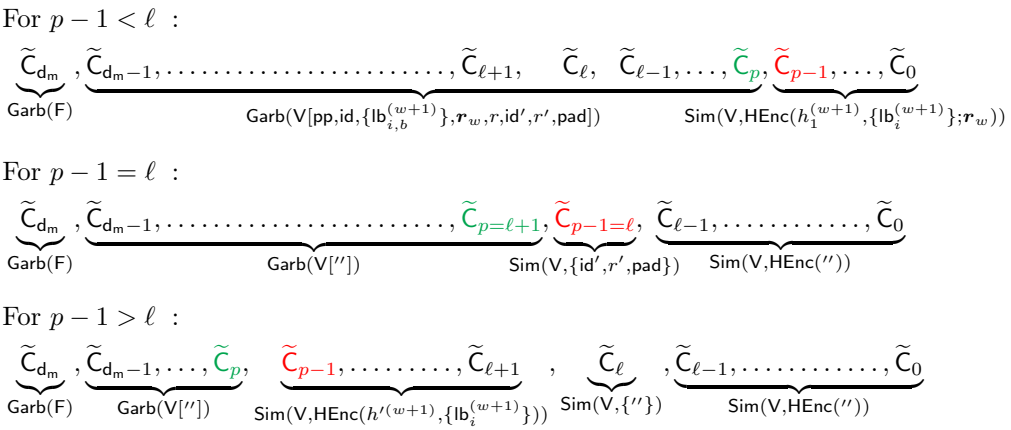

Fig. 7: Method of generating circuits in $\mathbf{Hyb}_{1.p}$ depending on the value of $p-1$ relative to the value of ℓ. Use of $h_1^{(w+1)}$ in HEnc on the LHS is from the assumption that pth has the leftmost leaf as an endpoint. $''$ is the ditto symbol.

Next we will prove $\mathbf{Hyb}_1 \stackrel{c}{\equiv} \mathbf{Hyb}_{1.\mathsf{d_m}}$. In $\mathbf{Hyb}_1$ (Fig. 6), all circuits are simulated. In $\mathbf{Hyb}_{1.\mathsf{d_m}}$ (Fig. 8), all circuits are simulated except for $\widetilde{\mathsf{C}}_{\mathsf{d_m}}$. The two hybrids are the same after constructing circuit $\widetilde{\mathsf{C}}_{\mathsf{d_m}}$ and its labels. So, either hybrid can be simulated by knowing r, the induced path pth, and the pair $(\widetilde{\mathsf{C}}_{\mathsf{d_m}}, \{\mathsf{lb}_i^{(\mathsf{d_m})}\})$. For ease of notation let $(\widetilde{\mathsf{C}}, \{\mathsf{lb}_i\})$ and $(\widetilde{\mathsf{C}}', \{\mathsf{lb}'_i\})$ denote the distribution of the tuple $(\widetilde{\mathsf{C}}_{\mathsf{d_m}}, \{\mathsf{lb}_i^{(\mathsf{d_m})}\})$ in $\mathbf{Hyb}_1$ and $\mathbf{Hyb}_{1.0}$, respectively. We have $(\widetilde{\mathsf{C}}, \{\mathsf{lb}_i\}) \stackrel{\$}{\leftarrow} \mathsf{Sim}(\mathsf{F}, \{\mathsf{id}', r'\})$ for random $\mathsf{id}', r' \stackrel{\$}{\leftarrow} \{0,1\}^\lambda$. In $\mathbf{Hyb}_{1.0}$, letting $\mathsf{C}_{\mathsf{d_m}} := \mathsf{F}[\mathsf{id}, \mathsf{id}', r, r']$ for random r, we have

$$\begin{aligned}(\widetilde{\mathsf{C}}', \{\mathsf{lb}_{i,b}\}) &\stackrel{\$}{\leftarrow} \mathsf{Garb}(\mathsf{C}_{\mathsf{d_m}})\\ \{\mathsf{lb}'_i\} &:= \{\mathsf{lb}_{i,z_{\mathsf{d_m}}[i]}\},\end{aligned}$$

where $z_{\mathsf{d_m}} \stackrel{\$}{\leftarrow} \{0,1\}^{3\lambda}$ if $\ell < \mathsf{d_m}$ and $z_{\mathsf{d_m}} := (\mathsf{Val}[v_1^{(\mathsf{d_m})}]^{\times\lambda}, x_1, x'_1)$ otherwise, where $\mathsf{Val}[v_1^{(\mathsf{d_m})}]^{\times\lambda} = 0^\lambda$. By simulation security of garbled circuits

$$(\widetilde{\mathsf{C}}', \{\mathsf{lb}'_i\}) \stackrel{c}{\equiv} \mathsf{Sim}(\mathsf{F}, \mathsf{C}_{\mathsf{d_m}}(z_{\mathsf{d_m}})) \stackrel{c}{\equiv} \mathsf{Sim}(\mathsf{F}, \{\mathsf{id}', r'\}).$$

If $\ell < \mathsf{d_m}$ and $z_{\mathsf{d_m}}$ is random, then $\mathsf{C}_{\mathsf{d_m}}(z_{\mathsf{d_m}}) = \{\mathsf{id}, r\}$ with probability $2^{-\lambda}$. If $\ell = \mathsf{d_m}$ and $z_{\mathsf{d_m}} := (0^\lambda, x_1, x'_1)$ then $\mathsf{C}_{\mathsf{d_m}}(z_{\mathsf{d_m}}) = \{\mathsf{id}', r'\}$ with probability 1. Thus, $(r, \mathsf{pth}, \widetilde{\mathsf{C}}, \{\mathsf{lb}_i\}) \stackrel{c}{\equiv} (r, \mathsf{pth}, \widetilde{\mathsf{C}}', \{\mathsf{lb}'_i\})$, proving $\mathbf{Hyb}_1 \stackrel{c}{\equiv} \mathbf{Hyb}_{1.0}$ and completing the proof of Lemma 6. □

Lemma 7 *For all $p \in \{0, \ldots, \mathsf{d_m} - 1\}$, $\mathbf{Hyb}_{1.p} \stackrel{c}{\equiv} \mathbf{Hyb}_{1.p+1}$.*

Proof. First, consider the circuits created in either hybrid:

$$\begin{aligned}\mathbf{Hyb}_{1.p} &: \overbrace{\widetilde{\mathsf{C}}_{\mathsf{d_m}}, \ldots, \widetilde{\mathsf{C}}_{p+1}, \widetilde{\mathsf{C}}_{p}}^{\mathsf{Garb}}, \overbrace{\widetilde{\mathsf{C}}_{p-1}, \ldots, \widetilde{\mathsf{C}}_{0}}^{\mathsf{Sim}}\\ \mathbf{Hyb}_{1.p+1} &: \underbrace{\widetilde{\mathsf{C}}_{\mathsf{d_m}}, \ldots, \widetilde{\mathsf{C}}_{p+1}}_{\mathsf{Garb}}, \underbrace{\widetilde{\mathsf{C}}_{p}, \widetilde{\mathsf{C}}_{p-1}, \ldots, \widetilde{\mathsf{C}}_{0}}_{\mathsf{Sim}}\end{aligned}$$

$\widetilde{\mathsf{C}}_{\mathsf{d_m}}, \{\mathsf{lb}_{i,b}^{(\mathsf{d_m})}\}, \ldots, \widetilde{\mathsf{C}}_{p+1}, \{\mathsf{lb}_{i,b}^{(p+1)}\}$ are the same in both hybrids. They differ only in the distribution of $(\widetilde{\mathsf{C}}_p, \{\mathsf{lb}_i^{(p)}\})$; it is generated honestly in $\mathbf{Hyb}_{1.p}$ and simulated in $\mathbf{Hyb}_{1.p+1}$. There are three possible ways $(\widetilde{\mathsf{C}}_p, \{\mathsf{lb}_i^{(p)}\})$ can be simulated in $\mathbf{Hyb}_{1.p+1}$ depending on the value of p relative to ℓ (see Fig. 7, but note that the figure illustrates $\mathbf{Hyb}_{1.p}$, not $\mathbf{Hyb}_{1.p+1}$). First, if $p < \ell$, then $(\widetilde{\mathsf{C}}_p, \{\mathsf{lb}_i^{(p)}\})$ is simulated using a hash encryption of

$\mathbf{Hyb}_{1.p}$:
$r, \mathsf{id}', r' \xleftarrow{\$} \{0,1\}^\lambda ; \mathsf{pad} \xleftarrow{\$} \{0,1\}^{2(n-1)} ; z_{\mathsf{d_m}} \xleftarrow{\$} \{0,1\}^{3\lambda}$
For all $0 \le w \le \mathsf{d_m}$ sample random $\boldsymbol{r}_w$
For i from $\mathsf{d_m}-1$ to $\ell+1$ do ▷ Generate "ghost" hash inputs for levels below pth
$v'^{(i)} \xleftarrow{\$} \{0,1\}^\lambda$; $z_i := (h'^{(i+1)}, h'^{(i+1)}, v'^{(i)})$
$(\widetilde{\mathsf{C}}_{\mathsf{d_m}}, \{\mathsf{lb}_{i,b}^{(\mathsf{d_m})}\}) \leftarrow^{\$} \mathsf{Garb}(\mathsf{F}[\mathsf{id}, \mathsf{id}', r, r']) ; \{\mathsf{lb}_i^{(\mathsf{d_m})}\} := \{\mathsf{lb}_{i,z_{\mathsf{d_m}}[i]}^{(\mathsf{d_m})}\}$
For w from $\mathsf{d_m}-1$ to p do $(\widetilde{\mathsf{C}}_w, \{\mathsf{lb}_{i,b}^{(w)}\}) \xleftarrow{\$} \mathsf{Garb}(\mathsf{V}[\mathsf{pp}, \mathsf{id}, \{\mathsf{lb}_{i,b}^{(w+1)}\}, \boldsymbol{r}_w, r, \mathsf{id}', r', \mathsf{pad}])$
$\{\mathsf{lb}_i^{(p)}\} := \{\mathsf{lb}_{i,z_p[i]}^{(p)}\}$ ▷ Final set of honest labels
If $p-1 \ge \ell$ then ▷ If circuits at, or below, pth leaf level are simulated
  If $p-1 > \ell$ then for $p-1$ to $\ell+1$ do ▷ Below pth leaf level
    $\{\mathsf{cth}_{i,b}^{(w+1)}\} \leftarrow \mathsf{HEnc}(\mathsf{pp}, h'^{(w+1)}, \{\mathsf{lb}_{i,b}^{(w+1)}\}; \boldsymbol{r}_w) ; (\widetilde{\mathsf{C}}_w, \{\mathsf{lb}_i^{(w)}\}) \xleftarrow{\$} \mathsf{Sim}(\mathsf{V}, \{\mathsf{cth}_{i,b}^{(w+1)}\})$
  $(\widetilde{\mathsf{C}}_\ell, \{\mathsf{lb}_i^{(\ell)}\}) \xleftarrow{\$} \mathsf{Sim}(\mathsf{V}, \{\mathsf{id}', r', \mathsf{pad}\})$ ▷ At pth leaf level
  For w from $\ell-1$ to 0 do ▷ From interior pth nodes to root
    $\{\mathsf{cth}_{i,b}^{(w+1)}\} \leftarrow \mathsf{HEnc}(\mathsf{pp}, h_1^{(w+1)}, \{\mathsf{lb}_i^{(w+1)}\}; \boldsymbol{r}_w) ; (\widetilde{\mathsf{C}}_w, \{\mathsf{lb}_i^{(w)}\}) \xleftarrow{\$} \mathsf{Sim}(\mathsf{V}, \{\mathsf{cth}_{i,b}^{(w+1)}\})$
Else ▷ If all circuits at, and below, pth leaf level are honest
  For w from $p-1$ to 0 do
    $\{\mathsf{cth}_{i,b}^{(w+1)}\} \leftarrow \mathsf{HEnc}(\mathsf{pp}, h_1^{(w+1)}, \{\mathsf{lb}_i^{(w+1)}\}; \boldsymbol{r}_w) ; (\widetilde{\mathsf{C}}_w, \{\mathsf{lb}_i^{(w)}\}) \xleftarrow{\$} \mathsf{Sim}(\mathsf{V}, \{\mathsf{cth}_{i,b}^{(w+1)}\})$
$\{\mathsf{cth}_{i,b}^{(0)}\} \xleftarrow{\$} \mathsf{HEnc}(\mathsf{pp}, \mathsf{h}_{\mathsf{root}}, \{\mathsf{lb}_i^{(0)}\})$
Return $\mathsf{m}_2 := (\widetilde{\mathsf{C}}_{\mathsf{d_m}}, \ldots, \widetilde{\mathsf{C}}_0, \{\mathsf{cth}_{i,b}^{(0)}\}, r)$

Fig. 8: $\mathbf{Hyb}_{1.p}$ for $0 \le p \le \mathsf{d_m}$. The last $p+1$ circuits in $\mathbf{Hyb}_{1.p}$ are generated honestly and the remainder are simulated. See Lemma 4.

$\{\mathsf{lb}_i^{(p+1)}\}$ with z_{p+1}. If $p = \ell$, then $(\widetilde{\mathsf{C}}_p, \{\mathsf{lb}_i^{(p)}\})$ is simulated using random output since $\mathsf{BP}(\mathsf{id}) = 0$. Finally, if $p > \ell$, then $(\widetilde{\mathsf{C}}_p, \{\mathsf{lb}_i^{(p)}\})$ is simulated using a hash encryption of $\{\mathsf{lb}_i^{(p+1)}\}$ using "ghost" value z_{p+1}. We will prove that in each of these three possible cases, it holds that $(\widetilde{\mathsf{C}}_p, \{\mathsf{lb}_i^{(p)}\})_{\mathbf{Hyb}_{1.p}} \overset{c}{\equiv} (\widetilde{\mathsf{C}}_p, \{\mathsf{lb}_i^{(p)}\})_{\mathbf{Hyb}_{1.p+1}}$.

1. If $p < \ell$:

$$\mathbf{Hyb}_{1.p} : \begin{cases} (\widetilde{\mathsf{C}}_p, \{\mathsf{lb}_{i,b}^{(p)}\}) \xleftarrow{\$} \mathsf{Garb}(\mathsf{V}[\mathsf{pp}, \mathsf{id}, \{\mathsf{lb}_{i,b}^{(p+1)}\}, \boldsymbol{r}_p, r, \mathsf{id}', r', \mathsf{pad}]) \\ \{\mathsf{lb}_i^{(p)}\} := \{\mathsf{lb}_{i,z_p[i]}^{(p)}\} \quad \text{where } z_p = (h_1^{(p+1)}, h_2^{(p+1)}, v_1^{(p)}) \end{cases}$$

$$\mathbf{Hyb}_{1.p+1} : \begin{cases} \{\mathsf{cth}_{i,b}^{(p+1)}\} \leftarrow \mathsf{HEnc}(\mathsf{pp}, h_1^{(p+1)}, \{\mathsf{lb}_i^{(p+1)}\}; \boldsymbol{r}_p) \\ (\widetilde{\mathsf{C}}_p, \{\mathsf{lb}_i^{(p)}\}) \xleftarrow{\$} \mathsf{Sim}(\mathsf{V}, \{\mathsf{cth}_{i,b}^{(p+1)}\}) \end{cases} \tag{7}$$

By simulation security of garbled circuits,

$$\begin{aligned} (\widetilde{\mathsf{C}}_p, \{\mathsf{lb}_i^{(p)}\})_{\mathbf{Hyb}_{1.p}} &\overset{c}{\equiv} \mathsf{Sim}(\mathsf{V}, \mathsf{C}_p(z_p)) \\ &\overset{c}{\equiv} \mathsf{Sim}(\mathsf{V}, \mathsf{HEnc}(\mathsf{pp}, h_1^{(p+1)}, \{\mathsf{lb}_{i,b}^{(p+1)}\}; \boldsymbol{r}_p)). \end{aligned} \tag{8}$$

The use of $h_1^{(p+1)}$ in Eq. 8 is due to the assumption that the path induced by id has the leftmost node as its terminal node. So by definition of C_p, its hardwired labels $\{\mathsf{lb}_{i,b}^{(p+1)}\}$ will be encrypted under $h_1^{(p+1)}$. Eq. 8 is identical to the RHS of Eq. 7, and thus when $p > \ell$ we have $(\widetilde{\mathsf{C}}_p, \{\mathsf{lb}_i^{(p)}\})_{\mathbf{Hyb}_{1.p}} \overset{c}{\equiv} (\widetilde{\mathsf{C}}_p, \{\mathsf{lb}_i^{(p)}\})_{\mathbf{Hyb}_{1.p+1}}$.

2. If $p = \ell$:

$$\mathbf{Hyb}_{1.p} : \begin{cases} (\widetilde{\mathsf{C}}_p, \{\mathsf{lb}_{i,b}^{(p)}\}) \xleftarrow{\$} \mathsf{Garb}(\mathsf{V}[\mathsf{pp}, \mathsf{id}, \{\mathsf{lb}_{i,b}^{(p+1)}\}, \boldsymbol{r}_p, r, \mathsf{id}', r', \mathsf{pad}]) \\ \{\mathsf{lb}_i^{(p)}\} := \{\mathsf{lb}_{i,z_p[i]}^{(p)}\} \quad \text{where } z_p = (0^\lambda, x_1, x_1') \end{cases}$$

$$\mathbf{Hyb}_{1.p+1}: \begin{cases} (\widetilde{\mathsf{C}}_p, \{\mathsf{lb}_i^{(p)}\}) \xleftarrow{\$} \mathsf{Sim}(\mathsf{V}, \{\mathsf{id}', r', \mathsf{pad}\}) \\ \text{where } \mathsf{id}', r' \xleftarrow{\$} \{0,1\}^\lambda ; \mathsf{pad} \xleftarrow{\$} \{0,1\}^{2(n-1)} \end{cases} \tag{9}$$

By simulation security of garbled circuits,

$$\begin{aligned} (\widetilde{\mathsf{C}}_p, \{\mathsf{lb}_i^{(p)}\})_{\mathbf{Hyb}_{1.p}} &\overset{c}{\equiv} \mathsf{Sim}(\mathsf{V}, \mathsf{C}_p(z_p)) \\ &\overset{c}{\equiv} \mathsf{Sim}(\mathsf{V}, \{\mathsf{id}', r', \mathsf{pad}\}) . \end{aligned} \tag{10}$$

When $p = \ell$, $z_p = (0^\lambda, x_1, x_1')$ which causes $\mathsf{C}_p(z_p)$ to output a random ID and signal string. So, Equation 10 is identical to the first line of Eq. 9. Thus if $p = \ell$, we have $(\widetilde{\mathsf{C}}_p, \{\mathsf{lb}_i^{(p)}\})_{\mathbf{Hyb}_{1.p}} \overset{c}{\equiv} (\widetilde{\mathsf{C}}_p, \{\mathsf{lb}_i^{(p)}\})_{\mathbf{Hyb}_{1.p+1}}$.

3. If $p > \ell$:

$$\mathbf{Hyb}_{1.p}: \begin{cases} (\widetilde{\mathsf{C}}_p, \{\mathsf{lb}_{i,b}^{(p)}\}) \xleftarrow{\$} \mathsf{Garb}(\mathsf{V}[\mathsf{pp}, \mathsf{id}, \{\mathsf{lb}_{i,b}^{(p+1)}\}, \boldsymbol{r}_p, r, \mathsf{id}', r', \mathsf{pad}]) \\ \{\mathsf{lb}_i^{(p)}\} := \{\mathsf{lb}_{i,z_p[i]}^{(p)}\} \quad \text{where } z_p = (h'^{(p+1)}, h'^{(p+1)}, v'^{(p)}) \end{cases}$$

$$\mathbf{Hyb}_{1.p+1}: \begin{cases} \{\mathsf{cth}_{i,b}^{(p+1)}\} \leftarrow \mathsf{HEnc}(\mathsf{pp}, h'^{(p+1)}, \{\mathsf{lb}_i^{(p+1)}\}; \boldsymbol{r}_p) \\ (\widetilde{\mathsf{C}}_p, \{\mathsf{lb}_i^{(p)}\}) \xleftarrow{\$} \mathsf{Sim}(\mathsf{V}, \{\mathsf{cth}_{i,b}^{(p+1)}\}) \\ \text{where } h'^{(p+1)} \leftarrow \mathsf{Hash}(\mathsf{pp}, z_{p+1}) \text{ is pseudorandom} \end{cases} \tag{11}$$

Consider evaluating $\widetilde{\mathsf{C}}_p$ on labels $\{\mathsf{lb}_{i,z_p[i]}^{(p)}\}$ as in $\mathbf{Hyb}_{1.p}$:

$$\begin{aligned} \mathsf{Eval}(\widetilde{\mathsf{C}}_p, \{\mathsf{lb}_{i,z_p[i]}^{(p)}\}) &= \mathsf{V}[\mathsf{pp}, \mathsf{id}, \{\mathsf{lb}_{i,b}^{(p+1)}\}, \boldsymbol{r}_p, r, \mathsf{id}', r', \mathsf{pad}](h'^{(p+1)}, h'^{(p+1)}, v'^{(p)}) \\ &= \begin{cases} \mathsf{HEnc}(\mathsf{pp}, h'^{(p+1)}, \{\mathsf{lb}_{i,b}^{(p+1)}\}; \boldsymbol{r}_p) & \text{if } \mathsf{id}[v_p'] = 0 \\ \mathsf{HEnc}(\mathsf{pp}, h'^{(p+1)}, \{\mathsf{lb}_{i,b}^{(p+1)}\}; \boldsymbol{r}_p) & \text{otherwise} \end{cases} \\ &= \mathsf{HEnc}(\mathsf{pp}, h'^{(p+1)}, \{\mathsf{lb}_{i,b}^{(p+1)}\}; \boldsymbol{r}_p) . \end{aligned} \tag{12}$$

Equation 12 is identical to the RHS of Eq. 11 (first), up to the labels $\{\mathsf{lb}_i^{(p+1)}\}$ in Eq. 11 vs. $\{\mathsf{lb}_{i,b}^{(p+1)}\}$ in Eq. 12. By simulation security, the labels $\{\mathsf{lb}_i^{(p+1)}\}$ in Eq. 11 are computationally indistinguishable from labels $\{\mathsf{lb}_{i,z_{p+1}[i]}^{(p+1)}\}$. Thus $\{\mathsf{lb}_{i,z_{p+1}[i]}^{(p+1)}\}_{\mathbf{Hyb}_{1.p}} \overset{c}{\equiv} \{\mathsf{lb}_i^{(p+1)}\}_{\mathbf{Hyb}_{1.p+1}}$. By HE semantic security, $\mathsf{HEnc}(\mathsf{pp}, h'^{(p+1)}, \{\mathsf{lb}_{i,b}^{(p+1)}\}; \boldsymbol{r}_p) \overset{c}{\equiv} \mathsf{HEnc}(\mathsf{pp}, h'^{(p+1)}, \{\mathsf{lb}_i^{(p+1)}\}; \boldsymbol{r}_p)$, and hence $(\widetilde{\mathsf{C}}_p, \{\mathsf{lb}_i^{(p)}\})_{\mathbf{Hyb}_{1.p}} \overset{c}{\equiv} (\widetilde{\mathsf{C}}_p, \{\mathsf{lb}_i^{(p)}\})_{\mathbf{Hyb}_{1.p+1}}$ when $p > \ell$, which completes the proof of Lemma 7. □

7.2 Proof of Lemma 5

Lemma 5 states that $\mathbf{Hyb}_1 \overset{c}{\equiv} \mathbf{Hyb}_2$. So, we must show that $\mathsf{m}_2 := (\widetilde{\mathsf{C}}_{\mathsf{d_m}}, \ldots, \widetilde{\mathsf{C}}_0, \{\mathsf{cth}_{i,b}^{(0)}\}, r)$, as sampled in $\mathbf{Hyb}_1$, is computationally indistinguishable from random. We will argue that each element of m_2 is pseudorandom.

First consider the circuit $\widetilde{\mathsf{C}}_{\mathsf{d_m}}$. It is formed as $(\widetilde{\mathsf{C}}_{\mathsf{d_m}}, \{\mathsf{lb}_i^{(\mathsf{d_m})}\}) \xleftarrow{\$} \mathsf{Sim}(\mathsf{F}, \{\mathsf{id}', r'\})$ where $\mathsf{id}', r' \xleftarrow{\$} \{0,1\}^\lambda$. Since the inputs id', r' are random, by anonymous security of garbled circuits the distribution $(\widetilde{\mathsf{C}}_{\mathsf{d_m}}, \{\mathsf{lb}_i^{(\mathsf{d_m})}\})$ is pseudorandom.

For w from $\mathsf{d_m} - 1$ to $\ell + 1$ the circuits are formed as $(\widetilde{\mathsf{C}}_w, \{\mathsf{lb}_i^{(w)}\}) \xleftarrow{\$} \mathsf{Sim}(\mathsf{V}, \{\mathsf{cth}_{i,b}^{(w+1)}\})$ where $\{\mathsf{cth}_{i,b}^{(w+1)}\} \xleftarrow{\$} \mathsf{HEnc}(\mathsf{pp}, h'^{(w+1)}, \{\mathsf{lb}_i^{(w+1)}\})$. $\{\mathsf{cth}_{i,b}^{(w+1)}\}$ is pseudorandom by anonymous semantic security of HE, and so by anonymous security of GS, $(\widetilde{\mathsf{C}}_w, \{\mathsf{lb}_i^{(w)}\})$ is also pseudorandom.

For $w = \ell$ we have $(\widetilde{\mathsf{C}}_\ell, \{\mathsf{lb}_i^{(\ell)}\}) \xleftarrow{\$} \mathsf{Sim}(\mathsf{V}, \{\mathsf{id}', r', \mathsf{pad}\})$ where $\mathsf{id}', r' \xleftarrow{\$} \{0,1\}^\lambda$, $\mathsf{pad} \xleftarrow{\$} \{0,1\}^{2(n-1)}$, so again by anonymous security of garbled circuits, the distribution $(\widetilde{\mathsf{C}}_\ell, \{\mathsf{lb}_i^{(\ell)}\})$ is pseudorandom.

For w from $\ell - 1$ to 0 we have $\{\mathsf{cth}_{i,b}^{(w+1)}\} \xleftarrow{\$} \mathsf{HEnc}(\mathsf{pp}, h_1^{(w+1)}, \{\mathsf{lb}_i^{(w+1)}\})$ and $(\widetilde{\mathsf{C}}_w, \{\mathsf{lb}_i^{(w)}\}) \xleftarrow{\$} \mathsf{Sim}(\mathsf{V}, \{\mathsf{cth}_{i,b}^{(w+1)}\})$. Where, again, the use of $h_1^{(w+1)}$ in HEnc is from the assumption on pth. For all w from $\ell - 1$ to

0, $\{\mathsf{cth}_{i,b}^{(w+1)}\}$ is pseudorandom by anonymous semantic security of HE, and thus by anonymous security of GS, $(\widetilde{\mathsf{C}}_w, \{\mathsf{lb}_i^{(w)}\})$ is also pseudorandom.

Next in m_2 is the ciphertext, which in $\mathbf{Hyb}_1$ is formed as $\{\mathsf{cth}_{i,b}^{(0)}\} \xleftarrow{\$} \mathsf{HEnc}(\mathsf{pp}, \mathsf{h}_{\mathsf{root}}, \{\mathsf{lb}_i^{(0)}\})$. $\{\mathsf{cth}_{i,b}^{(0)}\}$ is pseudorandom by anonymous semantic security of HE. The final element of m_2 is the signal string r, which is sampled uniformly at random. Hence m_2 is pseudorandom in the view of R, proving $\mathbf{Hyb}_1 \overset{c}{\equiv} \mathbf{Hyb}_2$ and completing the proof. □

Remark 1 *In the proofs above, we assumed that the path induced by evaluating* $\mathsf{BP}(\mathsf{id})$ *always travelled to the left child. In the general case, the path in Eq. 6 ending in* $v_1^{(\ell)}$ *just needs to be changed to the path induced by* $\mathsf{BP}(\mathsf{id})$ *ending in the appropriate leaf* $u \in T$*. The proofs should then be updated accordingly.*

Acknowledgements. S. Garg is supported in part by DARPA under Agreement No. HR00112020026, AFOSR Award FA9550-19-1-0200, NSF CNS Award 1936826, and research grants by Visa Inc, Supra, JP Morgan, BAIR Commons Meta Fund, Stellar Development Foundation, and a Bakar Fellows Spark Award. P. Miao is supported in part by the NSF CNS Award 2247352, a DPI Science Team Seed Grant, a Meta Award, and a Brown DSI Seed Grant. M. Hajiabadi is supported in part by an NSERC Discovery Grant 03270, and a Meta Research Award.

References

ABD+21. Navid Alamati, Pedro Branco, Nico Döttling, Sanjam Garg, Mohammad Hajiabadi, and Sihang Pu. Laconic private set intersection and applications. In Kobbi Nissim and Brent Waters, editors, *TCC 2021: 19th Theory of Cryptography Conference, Part III*, volume 13044 of *Lecture Notes in Computer Science*, pages 94–125, Raleigh, NC, USA, November 8–11, 2021. Springer, Heidelberg, Germany. 2, 3, 4, 5, 7, 8, 10

ALOS22. Diego F. Aranha, Chuanwei Lin, Claudio Orlandi, and Mark Simkin. Laconic private set-intersection from pairings. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022: 29th Conference on Computer and Communications Security*, pages 111–124, Los Angeles, CA, USA, November 7–11, 2022. ACM Press. 2, 3

BFK+09. Mauro Barni, Pierluigi Failla, Vladimir Kolesnikov, Riccardo Lazzeretti, Ahmad-Reza Sadeghi, and Thomas Schneider. Secure evaluation of private linear branching programs with medical applications. In Michael Backes and Peng Ning, editors, *Computer Security – ESORICS 2009*, pages 424–439, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. 2

BLSV18. Zvika Brakerski, Alex Lombardi, Gil Segev, and Vinod Vaikuntanathan. Anonymous IBE, leakage resilience and circular security from new assumptions. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part I*, volume 10820 of *Lecture Notes in Computer Science*, pages 535–564, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Heidelberg, Germany. 2, 3, 4, 7, 8

BPSW07. Justin Brickell, Donald E. Porter, Vitaly Shmatikov, and Emmett Witchel. Privacy-preserving remote diagnostics. In Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson, editors, *ACM CCS 2007: 14th Conference on Computer and Communications Security*, pages 498–507, Alexandria, Virginia, USA, October 28–31, 2007. ACM Press. 2

CDG+17. Chongwon Cho, Nico Döttling, Sanjam Garg, Divya Gupta, Peihan Miao, and Antigoni Polychroniadou. Laconic oblivious transfer and its applications. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part II*, volume 10402 of *Lecture Notes in Computer Science*, pages 33–65, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Heidelberg, Germany. 2, 3, 8

CDPP22. Kelong Cong, Debajyoti Das, Jeongeun Park, and Hilder V. L. Pereira. SortingHat: Efficient private decision tree evaluation via homomorphic encryption and transciphering. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022: 29th Conference on Computer and Communications Security*, pages 563–577, Los Angeles, CA, USA, November 7–11, 2022. ACM Press. 2

CGH+21. Melissa Chase, Sanjam Garg, Mohammad Hajiabadi, Jialin Li, and Peihan Miao. Amortizing rate-1 OT and applications to PIR and PSI. In Kobbi Nissim and Brent Waters, editors, *TCC 2021: 19th Theory of Cryptography Conference, Part III*, volume 13044 of *Lecture Notes in Computer Science*, pages 126–156, Raleigh, NC, USA, November 8–11, 2021. Springer, Heidelberg, Germany. 2, 12

DG17. Nico Döttling and Sanjam Garg. Identity-based encryption from the Diffie-Hellman assumption. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part I*, volume 10401 of *Lecture Notes in Computer Science*, pages 537–569, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Heidelberg, Germany. 2, 3, 4, 7

DGGM19. Nico Döttling, Sanjam Garg, Vipul Goyal, and Giulio Malavolta. Laconic conditional disclosure of secrets and applications. In David Zuckerman, editor, *60th Annual Symposium on Foundations of Computer Science*, pages 661–685, Baltimore, MD, USA, November 9–12, 2019. IEEE Computer Society Press. 2, 5

DGI+19. Nico Döttling, Sanjam Garg, Yuval Ishai, Giulio Malavolta, Tamer Mour, and Rafail Ostrovsky. Trapdoor hash functions and their applications. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part III*, volume 11694 of *Lecture Notes in Computer Science*, pages 3–32, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Heidelberg, Germany. 2, 3

DKL+23. Nico Döttling, Dimitris Kolonelos, Russell W. F. Lai, Chuanwei Lin, Giulio Malavolta, and Ahmadreza Rahimi. Efficient laconic cryptography from learning with errors. In *Advances in Cryptology – EUROCRYPT 2023, Part III*, Lecture Notes in Computer Science, pages 417–446. Springer, Heidelberg, Germany, June 2023. 3

Gen09. Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st Annual ACM Symposium on Theory of Computing*, pages 169–178, Bethesda, MD, USA, May 31 – June 2, 2009. ACM Press. 1

GGH19. Sanjam Garg, Romain Gay, and Mohammad Hajiabadi. New techniques for efficient trapdoor functions and applications. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019, Part III*, volume 11478 of *Lecture Notes in Computer Science*, pages 33–63, Darmstadt, Germany, May 19–23, 2019. Springer, Heidelberg, Germany. 7

GHMR18. Sanjam Garg, Mohammad Hajiabadi, Mohammad Mahmoody, and Ahmadreza Rahimi. Registration-based encryption: Removing private-key generator from IBE. In Amos Beimel and Stefan Dziembowski, editors, *TCC 2018: 16th Theory of Cryptography Conference, Part I*, volume 11239 of *Lecture Notes in Computer Science*, pages 689–718, Panaji, India, November 11–14, 2018. Springer, Heidelberg, Germany. 4

GHO20. Sanjam Garg, Mohammad Hajiabadi, and Rafail Ostrovsky. Efficient range-trapdoor functions and applications: Rate-1 OT and more. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020: 18th Theory of Cryptography Conference, Part I*, volume 12550 of *Lecture Notes in Computer Science*, pages 88–116, Durham, NC, USA, November 16–19, 2020. Springer, Heidelberg, Germany. 2

GRS22. Gayathri Garimella, Mike Rosulek, and Jaspal Singh. Structure-aware private set intersection, with applications to fuzzy matching. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology – CRYPTO 2022, Part I*, volume 13507 of *Lecture Notes in Computer Science*, pages 323–352, Santa Barbara, CA, USA, August 15–18, 2022. Springer, Heidelberg, Germany. 3, 14

GSW13. Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 75–92, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Heidelberg, Germany. 1

IP07. Yuval Ishai and Anat Paskin. Evaluating branching programs on encrypted data. In Salil P. Vadhan, editor, *TCC 2007: 4th Theory of Cryptography Conference*, volume 4392 of *Lecture Notes in Computer Science*, pages 575–594, Amsterdam, The Netherlands, February 21–24, 2007. Springer, Heidelberg, Germany. 1, 2, 6

KNL+19. Ágnes Kiss, Masoud Naderpour, Jian Liu, N. Asokan, and Thomas Schneider. SoK: Modular and efficient private decision tree evaluation. *Proceedings on Privacy Enhancing Technologies*, 2019(2):187–208, April 2019. 2

QWW18. Willy Quach, Hoeteck Wee, and Daniel Wichs. Laconic function evaluation and applications. In Mikkel Thorup, editor, *59th Annual Symposium on Foundations of Computer Science*, pages 859–870, Paris, France, October 7–9, 2018. IEEE Computer Society Press. 2, 3

Rab81. Michael O. Rabin. How to exchange secrets with oblivious transfer. Technical report, TR-81, Aiken Computation Lab, Harvard University, 1981. 1

Rab05. Michael O. Rabin. How to exchange secrets with oblivious transfer. Cryptology ePrint Archive, Report 2005/187, 2005. https://eprint.iacr.org/2005/187. 1

Yao86. Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th Annual Symposium on Foundations of Computer Science*, pages 162–167, Toronto, Ontario, Canada, October 27–29, 1986. IEEE Computer Society Press. 1

A Supplementary Figures

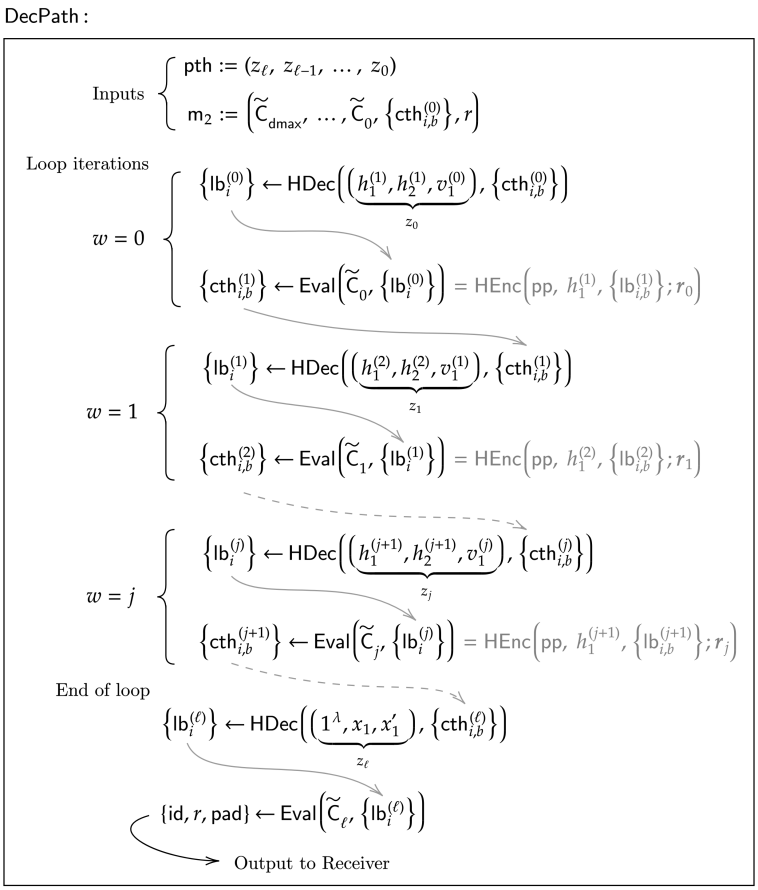

Fig. 9: Illustration of the progression of DecPath given in Figure 3, assuming the input path has endpoint the leftmost leaf and has value 1^λ.

Procedure $\mathsf{SetBP}^*(S, b_{\mathsf{pth}})$**:**

$x \leftarrow S$; $\lambda \leftarrow |x|$; $V, E, T \leftarrow \emptyset$

$\overline{\mathsf{WC}} \leftarrow \{i \mid x[i] \neq *\}$; $\overline{k} \leftarrow |\overline{\mathsf{WC}}|$ ▷ ascending ordered set of all non-wildcard indices

If $x[1] \neq *$ then $V \leftarrow V \cup \{v_\epsilon\}$; $\mathsf{Val}[v_\epsilon] \leftarrow 1$ ▷ root node if x doesn't start with *

If $x[1] = *$ then $V \leftarrow V \cup \{v_\epsilon\}$; $\mathsf{Val}[v_\epsilon] \leftarrow \overline{\mathsf{WC}}[1]$ ▷ root node if x starts with *

For $1 \leq i \leq \overline{k}$ do ▷ for every non-wildcard index of x

  $j \leftarrow \overline{\mathsf{WC}}[i]$; $a \leftarrow x[1..j]$; $V \leftarrow V \cup \{v_a\}$

  If $i = \overline{k}$ then $\mathsf{Val}[v_a] \leftarrow b_{\mathsf{pth}}$; $T \leftarrow T \cup \{v_a\}$ ▷ accept leaf

  Else $\mathsf{Val}[v_a] \leftarrow \overline{\mathsf{WC}}[i+1]$

  $a_{prev} \leftarrow x[1..\overline{\mathsf{WC}}[i-1]]$ ▷ previous interior node (If $i = 1$ then $a_{prev} \leftarrow \epsilon$)

  $E \leftarrow E \cup \{(v_{a_{prev}}, v_a, x[j])\}$ ▷ edge labelled with value of current non-* bit

  $a' \leftarrow x[1..(j-1)] \,\|\, (1 - x[j])$ ▷ a' is equal to a with the last bit flipped

  $V \leftarrow V \cup \{v_{a'}\}$; $T \leftarrow T \cup \{v_{a'}\}$; $\mathsf{Val}[v_{a'}] \leftarrow 1 - b_{\mathsf{pth}}$ ▷ reject leaf

  $E \leftarrow E \cup \{(v_{a_{prev}}, v_{a'}, 1 - x[j])\}$ ▷ edge with flipped value of current non-* bit

Return (V, E, T, Val)

Fig. 10: Annotated version of Fig. 5. Procedure for constructing a branching program from a singleton set containing a λ-bit string with wildcards. See Construction 2 and Section 5.3.